Component-based Grid Programming
Using the HOC-Service Architecture

Sergei Gorlatch

University of Münster, Germany
PARALLEL AND DISTRIBUTED COMPUTING: TOWARD GRIDS

Demand – growing:

- **Grand Challenges**: Scientific computing (numeric simulations in climate research, aircraft construction, biology etc.)
- Distributed & cooperative Internet applications (Games, E-learning, E-commerce)
PARALLEL AND DISTRIBUTED COMPUTING: TOWARD GRIDS

Demand – growing:

→ *Grand Challenges*: Scientific computing (numeric simulations in climate research, aircraft construction, biology etc.)
→ Distributed & cooperative Internet applications (Games, E-learning, E-commerce)

Conditions – improving:

→ Processors and networks more powerful and cheaper
→ Parallelism on multiple levels:
  - Inside a processor: pipelining, look-ahead, etc.
  - Across multiple processors of a PCs and parallel computers
  - Across multiple computers (Internet, *computational grids*)
PARALLEL AND DISTRIBUTED COMPUTING: TOWARD GRIDS

Demand – growing:

→ *Grand Challenges*: Scientific computing (numeric simulations in climate research, aircraft construction, biology etc.)
→ Distributed & cooperative Internet applications (Games, E-learning, E-commerce)

Conditions – improving:

→ Processors and networks more powerful and cheaper
→ Parallelism on multiple levels:
  • Inside a processor: pipelining, look-ahead, etc.
  • Across multiple processors of a PCs and parallel computers
  • Across multiple computers (Internet, *computational grids*)

Software – advancing?

+ Standards (MPI, Java+RMI, etc.) ⇒ Portability across platforms
- Low-level, cumbersome, error-prone
Compute-intensive applications are distributed among multiple high-performance computers connected via the Internet.

Application programming is supported by so-called grid middleware (recent standard: *Globus Toolkit*)
Low-level programming model remains a challenge in the HPC world (e.g., MPI with explicit send-recv).

Difficult to manage large numbers of processors.

Grids connect several HPC systems together, thereby making the task of application programmers even more challenging.

Grid-specific tasks are added:

- heterogeneous processing nodes
- heterogeneous, dynamic interconnects

Grid middleware deals with these tasks:

- Grid Services extend Web Services to become stateful and transient
- Web Services allow for remote procedure calls via the Internet (passing proxies and firewalls), using SOAP via HTTP
- Middleware should take care of resources
- What should the programmer take care of?
PROGRAMMER’S TASKS IN GLOBUS

- Configure services & resources using an extended WSDL format
- Map service namespaces to implementation packages
- Implement services, resources and "homes" (factories)
- Write WSDD deployment configuration
- Deploy the Grid Application Archive (JNDI)
AN EXAMPLE WSDL-INTERFACE FOR THE GLOBUS WSRF

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="MasterService"
    targetNamespace="http://org.gridhocs/Master"
    xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <wsdl:import location="../../wsrf/properties/WS-ResourceProperties.wsdl"/>
    <wsdl:types>
        <schema targetNamespace="http://org.gridhocs/Master"
            xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:impl="http://impl.org/">
            <complexType name="ArrayOf_xsd_double">
                <complexContent>
                    <restriction base="soapenc:Array">
                        <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:double[]"/>
                    </restriction>
                </complexContent>
            </complexType>
        </schema>
    </wsdl:types>
    <wsdl:message name="configureRequest">
        <wsdl:part name="in0" type="impl:ArrayOf_xsd_string"/>
    </wsdl:message>
    <wsdl:message name="configureResponse">
        <part name="parameters" element="impl:void"/>
    </wsdl:message>
    <wsdl:portTypename="MasterPortType"
        wsdlpp:extends="wsrpw:GetResourceProperty
            wsrlw:ImmediateResourceTermination"
        wsrl:ResourceProperties="tns:MasterResourceProperties">
        <wsdl:operation name="configure" parameterOrder="in0">
            <wsdl:input message="impl:configureRequest"/>
            <wsdl:output message="impl:configureResponse"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```
**Approach: Grid Programming with HOCs**

- **HOC = Higher-Order Component**
  - Captures a typical pattern of parallel/distributed behaviour, with application-specific codes as parameters.

![Diagram showing HOC component repository with Map HOC, Pipeline HOC, Farm HOC, DWT HOC, and Divide & Conquer HOC.]

An Application Developer selects a component from the Component Repository and writes application-specific code, e.g., worker parameter for an image processing farm.
Components are developed by Grid experts capable of writing efficient parallel code for the target machines.

The component repository is packaged with necessary configuration files (WSDL, WSDD) in a GAR file deployed remotely.

- Repository of HOCs
  - Map HOC
  - Farm HOC
  - Pipeline HOC
  - Scan HOC
  - Divide & Conquer HOC

stored in GAR contains: GWSDL, GT3-Code, GWSDD

Application Developer

write mobile code parameter

Grid Expert

provides
GRID PROGRAMMING USING HOCs

Example: For a farm application the application programmer provides two parameters: Master and Worker.

A distributed implementation incl. all needed configuration files is provided as Grid Services by the component developer.
AN EXAMPLE: FARM-HOC EVALUATION

① client obtains Farm Service instance, ② Farm service obtains worker services ③, distributes calculations ④ and reassembles result ⑤
HOC-SA: SERVICE ARCHITECTURE FOR HOCs

→ A framework of HOC implementations in the form of grid-enabled Web services (incl. the configuration required for their deployment onto the established grid middleware)

→ programming by selection, composition and parametrization:

- code parameter
  - provided by the client
  - application specific code only

- server-sided implementation
  - distributed, parallel algorithm
  - efficient and architecture-tuned
    generic, i.e., not application specific
The code service & class loader of the HOC-SA

The code service & class loader of the HOC-SA

client

local code

... HOC1(Aid, Bid) ...

... code for A

code for B

server X

code service

remote class loader

server Y

instantiation of HOC1

service container

A B

HOC1

HOC2

A B C

grid
USING THE HOC-SA FOR IMAGE FILTERING

Application Client

Worker Hosts

split

process

overlapping subimages

Master Host

send image

recombine

retrieve result

Application Client
APIs and Developer roles in the HOC-SA

HOC-SA Component Repository

- Farm-HOC
- Reduce-HOC
- Divide & Conquer HOC

Component Framework
- Interfaces
- Service Definitions
- Configuration

Client API
- Portal
  - request service & send parameters
  - get Result

Service API
- Class Loader
- Code Service

Application Programmer

Component Developer

get Result

deploy

derive

HOC
case study: computing julia sets with the hoc-sa

- calculating fractal images is a compute intensive task
- procedure can be applied to multiple independent tiles
- straightforward parallelization possible
- dynamic process has varying time costs
for all experiments the distance between the server running the master code unit and the servers running the workers was ca. 500 kilometers

<table>
<thead>
<tr>
<th>1 remote server</th>
<th>2 remote servers</th>
<th>3 remote servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 processors</td>
<td>4 + 8 processors</td>
<td>4 + 8 + 12 processors</td>
</tr>
<tr>
<td>198,212 sec</td>
<td>128,165 sec</td>
<td>48,377 sec</td>
</tr>
</tbody>
</table>

Experiments observations:

- sequential time for a local evaluation was more than 1000 seconds (more than five times higher than using a remote server with 4 processors)
- transferring the result via SOAP takes much time (about 60 sec), due to the complexity of the SOAP encoding.
Performance prediction for HOCs

Application

Code parameter

Data

Client

Bytecode analysis

Runtime prediction/scheduling

Execution

HOC runtime prediction/server characteristics

Data/code parameter

Composition

Result

Internet

HOC1

HOC2
A Scalability Model for Computer Games

- real-time games (player positions etc.) are updated step-by-step

\[ S_{i-2} \quad S_{i-1} \quad S_i \quad S_{i+1} \quad S_{i+2} \]

\[ \phi_{S_{i-3}, S_{i-2}} \quad \phi_{S_{i-2}, S_{i-1}} \quad \phi_{S_{i-1}, S_i} \quad \phi_{S_i, S_{i+1}} \]
A Scalability Model for Computer Games

- Real-time games (player positions etc.) are updated step-by-step

$$S_{i-2} \rightarrow S_{i-1} \rightarrow S_i \rightarrow S_{i+1} \rightarrow S_{i+2}$$

- Analytical model for the maximum number of clients

![Graph showing supported clients vs. number of proxy servers](image1)

![Graph showing kBytes/sec vs. number of clients](image2)
EXAMPLE: COMPUTATION OVERLOAD

http://www.clanvikings.org/tnse/ustuff.html
EXAMPLE: COMPUTATION WITHIN THE TIME LIMIT

http://www.clanvikings.org/tnse/ustuff.html

Unreal Tournament © Epic Games
CONCLUSION

- The HOC-Service Architecture (HOC-SA) simplifies grid application development considerably.
- Many recurring patterns of parallel computation can be implemented as HOCs (Pipes, Divide & Conquer, ...).
- HOCs allow for communication across the boundaries of heterogeneous hardware and software in a grid-aware manner using standards like Web services and Globus/WSRF.
- The abstraction offered by HOCs does not cause a significant performance loss.
- The higher-order programming model allows for accurate performance prediction.