GRIDL: High-Performance and Distributed Interactive Data Language

Svetlana Shasharina, Ovsei Volberg, Peter Stoltz and Seth Veitzer

Tech-X Corporation

HPDC 2005, July 25, 2005
Poster
Overview

• Interactive Data Language
• Design of parallel IDL on a grid
• Design of IDL clients for Web/Grid Service
• Status
• Conclusions
Interactive Data Language is a powerful visualization and analysis tool

- 4GL: scripting and high-level (built on C)
- Well-suited for N-dimensional arrays
- Interworks with C/C++, Fortran, and Java
- Well established in scientific community
  - Fusion community uses IDL interface to access and analyze experimental data.
  - Plasma physics simulations (VORPAL: 3D code) make 2D plots, 3D vis and movies using IDL
  - Volumetric MRI data (Los Alamos biophysics and co-registration codes at Ohio Kettiring center)
  - Earth Sciences
Scientific computing is becoming parallel and distributed

- 3D simulations are large and need multiple processors (possibly distributed if latency allows): high-performance in a regular sense and on a grid
- Data (computational and experimental) is distributed and large (terabytes produced in remote experiments and simulations)
- Data analysis is distributed and massive
- Many levels of parallelisms and interactivity:
  - Some directives need to be parallel (data analysis)
  - Some should happen on one processor in an interactive mode: finalization of data analysis
Solutions (high-performance computing and Grids) do not address IDL

- **Grids:**
  - **Globus:**
    - GT3 (and pre GT3): secure running remote jobs
    - GT4: Web Services for scientific computing
  - **Web Services:**
    - Distributed client-server using Web technologies

- **MPI:** parallel computing

- **Grids and MPI do not address 4GLs and IDL in particular**
GRIDL: merging parallel distributed computing with IDL

• Parallel IDL
  – Running parallel IDL applications on clusters, supercomputers and grids

• IDL clients for Web Services

• 6 months of prototype work funded by DOE so far
Parallel IDL uses external C

- Dynamically Loadable Modules allow IDL to call external C functions by wrapping them.
- Wrapping C implementation of MPI exposes MPI in IDL
Parallel IDL on a Grid design uses external C and MPICH-G2

- MPICH-G2: MPICH using Globus underneath
- Wrapping MPICH now allows to run parallel IDL on a Grid
IDL client wraps generated stub (C or Java)
Status

• DLMaker: prototype tool for automation of IDL wrapping of external C code
• Testing parallel IDL on a simple grid
• Playing with gSoap (wrapping of C++ clients of gSoap Web Services into IDL)
• Playing with Globus (wrapping Java clients of GT3.2 Web Services using Java-IDL bridge)
DLMaker works for simple types

- **Input**: .h file with C signatures
  ```c
  extern "C" {
  void pro(double ,int, float);
  int func(int, float);
  }
  ```
- **Output**: what is needed for exposing these C functions into IDL (C wrapper, registration code, DLM description). Allows:
  ```idl
  pro testmodule
  pro, 2, 3, 1
  y = func3(1, 3)
  end
  ```
- **Limitations**:
  - No notion of in and out variables (only in)
  - No notion of pointers
Need more sophisticated DLMaker capable of:

- In, out variables for MPI calls with correct memory allocation
- Produce IDL keywords (some arguments should go there: types)
- Specializing to particular web services compiler (wrappers should deal with return variables and static variables correctly - specific!)
Simple MPI wrapper can be produced by our DLMaker

```c
static IDL_VPTR IDL_MPI_COMM_SIZE
(int argc, IDL_VPTR argv[]){
    IDL_VPTR tmp = IDL_Gettmp();
    int size;
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    tmp->type = IDL_TYP_LONG;
    tmp->value.l = size;
    return tmp;
}
```
Parallel IDL looks like MPI then

```idl
pro testmodule, x
    rank=mpi_comm_rank()
    nproc=mpi_comm_size()
    help, rank
    help, nproc
end
```
Parallel IDL needs a C driver (and some thinking about how to pass arguments to the main program)

```cxx
// startMPI.cxx
int main(int argc, char** argv) {
    int errorValue;
    MPI_Init(&argc,&argv);
    errorValue = IDL_Init(IDL_INIT_QUIET, &argc, argv);
    errorValue = IDL_ExecuteStr("testmodule, 1");
    MPI_Finalize();
    errorValue = IDL_Cleanup(0);
    return errorValue;
}
```

-------------------

```bash
mpirun -np 4 -v startMPI
```
Experiment with a grid (LAN) with 2-processors nodes (MPICH-G2 v. 1.2.6 and Globus 3.2, gcc3.43 and IDL v.6.1)
IDL clients for gSoap Web Service needed hand-wrapping

**Interface:**

```c
int wsidlhello__simprocl(int x, struct
    wsidlhello__simproclResponse {} *out);
```

**Wrapper:**

```c
void IDL_simpro1(int argc, IDL_VPTR* argv){
    int x = argv[0]->value.i;
    // gSoap runtime environment
    struct soap soap;
    // initialize it
    soap_init(&soap);
    float result;
    // Try to call the method
    soap_call_wsidlhello__simprocl(&soap, server, NULL, x,0);
    // Clean up
    soap_end(&soap);
}
```
Working with Java clients and using Java-IDL bridge was straightforward

- GT3.2
- IDL 6.1 (or higher)
- Set paths as needed
- IDL client mirroring Java client
Java client:

```java
public class Client{
    public static void main(String[] args){
        // Get command-line arguments
        URL GSH = new java.net.URL(args[0]);
        int a = Integer.parseInt(args[1]);
        // Get a reference to the MathService instance
        MathServiceGridLocator mathServiceLocator = new MathServiceGridLocator();
        MathPortType math = mathServiceLocator.getMathServicePort(GSH);
        // Call remote method 'add'
        math.add(a);
        // Get current value through remote method 'getValue'
        int value = math.getValue();
    }
}
```
IDL client creates java objects and uses the bridge to delegate to them:

pro CLIENT
; URL for the service
  gsh = OBJ_NEW("IDLJavaObject$URL", "java.net.URL", 
      "http://64.240.154.9:8090/ogsa/services/MathService")
; Service Locator
  mathLoc = OBJ_NEW("IDLJavaObject$LOC", 
      "com.txcorp.stubs.MathService.service.MathServiceGridLocator")
; Get the port using service locator and service handle
  math = mathLoc->getMathServicePort(gsh)
; Invoke the method
  math->add, 7
  res = math->getValue()
; Clean
  OBJ_DESTROY, gsh
  OBJ_DESTROY, mathLoc
end
Conclusions

• Need more powerful tool for generation of IDL wrappers from C descriptions (effective MPI and WSDL wrapper for C Web Services).
• Need to wrap more MPI routines.
• GT4 working with MPICH-g2
• Could use IDL-Java bridge to create IDL clients for Java Web Services. Works really well.
• Other 4GLs?
• Currently need new funding :-)

GRIDL, Tech-X, HPDC2005