Grid Computing 7700
Fall 2005
Lecture 6 and 8: Grid Programming Models

Gabrielle Allen
allen@bit.csc.lsu.edu

http://www.cct.lsu.edu/~gallen
Quiz #2

- What is the “jobmanager” in GRAM responsible for?
- Name three advanced features of GridFTP which are not present in the FTP protocol.
- Draw a diagram to show how third party data transfer works, show both the control and data channels.
- Draw a diagram to show the architecture of MDS as implemented with GRIS and GIIS.
- What does LDAP stand for?
Requirements

- Enable applications to make use of parallel, distributed, heterogeneous and changing Grid environments
- Applications themselves are often “legacy applications”, but at the cutting edge we have complex, dynamic and self-adaptive applications, and new application scenarios
- Learn from wealth of previous and current work in distributed and parallel computing
- Deal with new challenges, e.g. latency, concurrency, partial failures, unreliable services
Programming Models

- Developing applications
  - Compilers, OSs,
  - APIs, SDKs, Toolkits
  - Grid libraries ("numerical recipes in the Grid")

- Supporting tools
  - Make
  - Debuggers
  - Profilers

- Deploying applications
  - "mpirun"
  - Portals
  - Visual Studio for the Grid
What is a Programming Model

- Basically an interface separating "high-level" properties from "low-level" ones.
- Abstract machine, providing operations to the programming level above, and requiring implementations on all appropriate architectures below.
- The programming model should be abstract, but to be useful as a model is must address both abstraction (CS research) and effectiveness ... that is it’s implementation (real world applicability).
Why Use a Programming Model?

- **Abstraction**
  - Simplifies the structure of software implementations and makes it easier to design and construct.

- **Stability**
  - Provides standard interfaces which are stable over long time periods
  - Provides fixed requirements for implementation
  - Separation between higher level software developers and lower level implementers

- **Examples at CCT**
  - Cactus, Grid Application Toolkit, SAGA, GridSphere
General model requirements include (Models and Languages for Parallel Computation, Skillicon and Talia, 1996)
- Easy to program (model must hide unnecessary details from programmers, model should provide natural environment for programmers)
- Software development methodology
- Architecture independent (Grids are heterogeneous too)
- Easy to understand and teach (Large numbers of people have to be able to learn it ... shallow learning curve ideal)
- Effectively implementable (otherwise what use is it)
- Cost measures (execution time, cost of development and support, software engineering, is it worth the effort?)
What Kind of Model

- Abstract high level
  - Easy to build programs for developers
  - Harder to compile efficient code
- Low level models
  - Hard to build programs for developers
  - Easier to implement efficiently (if you know what you are doing)
Parallel Programming Models

- PVM
- MPI
- OpenMP
- Global Arrays
- HPF
- SHMEM
Grid Programming Issues

- Everything from parallel computing
- Portability
  - Architecture independence (virtual machines, prestaged executables and environments, executable repositories)
- Implementation interoperability (open protocols, services, APIs and SDKs)
- Adaptivity (reconfigure to changing environment, e.g. cache size, file space)
- Discovery (locate services and discover how to use them)
- Performance and QoS (performance models and contracts)
- Fault tolerance (partial failure, full failure, unreliable file transfer)
- Security (multiple sites, hierarchy of tasks, delegation of control, collaboration)
- Meta-models (grid compilers, OSs, APIs and SDKs)
- Latency
- Distributed memory
- Scientific computing (large scale, data types, data size)
Application Programming Interface (API)

- Wikipedia: “An application programming interface (API) is a set of definitions of the ways one piece of computer software communicates with another. It is a method of achieving abstraction, usually (but not necessarily) between lower-level and higher-level software.”
- Refers to definition, not implementation (although we often muddle this)
  - E.g., GAT API, Globus API, MPI, Google API,
- Often a language specific specification for a set of routines to facilitate application development
  - Routine name, number, order and type of arguments; mapping to language construct
- Good practice to always look for and provide fixed and well documented APIs
Software Development Kit (SDK)

- Wikipedia: “A software development kit (SDK), is typically a set of development tools that allows a software engineer to create applications for a certain software package, software framework, hardware platform, computer system, operating system or similar. “
- Often an implementation of an API
- Usually includes debugging tools, documentation, examples etc.
- Examples: MPICH, Globus Toolkit, Cactus Computational Toolkit
Current Grid Programming Models

1. Shared State Models (JavaSpaces, Publish/Subscribe)
2. Message Passing Models (MPI)
3. RPC and RMI Models (GridRPC, Java RMI)
4. Hybrid Models (OpenMP/MPI, OmniRPC, MPJ)
5. Peer-to-Peer (JXTA)
6. Grid APIs (Globus Toolkit, GAT, SAGA)
7. Application Frameworks (Cactus, GridSphere)
8. Component Models (CORBA, CoG, Legion)
9. Web Services Model (OGSA, Web Services)
Which Model is Best

- Still a matter of active research
- Time consuming and a lot of energy needed to test with real applications
- Will be application dependent (e.g. embarrassing parallel, loosely coupled, tightly coupled, data intensive, compute intensive, ...)
- High level models with abstract APIs and interfaces will help insulate application developers, and can then integrate with other models below.
1. Shared State Models

- Sharing of objects and data between machines
- Shared filesystems and memory
- Typically for shared memory machines or distributed machines with fast interconnect
- Programming models for the grid/distributed computing based on shared state where producers and consumers are decoupled.
1. Shared State Examples

- **JavaSpaces**
  - Java implementation of Linda tuplespace (tuples are represented as serialized objects)
  - Java for interoperability
  - Application viewed as processes which communicate by putting and getting objects into shared and persistent networked-spaces.
  - “put”, “take”, “read”
  - Like a shared data repository (CVS for applications)

- **Publish/subscribe**
  - Requires associative matching
2. Message Passing Models

- Processes run in disjoint address spaces and exchange information via messages
- More emphasis on the programmer doing the right thing --- advantages and disadvantages
- Heavily used on single machines in parallel processing
- Explicitly marshalled and static arguments
2. Message Passing Examples

- **MPI**
  - Message Passing Interface (MPI) is a standard API that defines two-sided messaging
  - Matched sends and receives
  - Many implementations, LAM, MPICH, vendor-specific
  - MPICH-G2 is grid-enabled implementation which can couple multiple machines (of different architectures)
  - TCP for inter-machine messaging and vendor-MPI for intra-machine messaging
  - Requires Globus for authentication and program initiation
  - Cactus work on showing how applications can be written to use MPICH-G2 efficiently on WANs.
  - Others: MagPIe (optimized collective operations), PACX-MPI
2. Message Passing Examples

- **One sided messaging**
  - Send operation does not have to have a receive operation
  - Supports irregular and asynchronous communication patterns
  - E.g. MPI-2 and Nexus
3. RPC and RMI Models

- Remote Procedure Call and Remote Method Invocation
  - Provide capabilities to invoke functions on remote machines, somewhat like message passing but more flexible operations and messages.
- Client/Server architecture
- Many different implementations, with different (and often incompatible) RPC protocols.
- To allow servers to be accessed by different clients, some standardized RPC systems are available:
  - Use an interface description language (IDL)
  - Additional features such as errors and recovery
- Examples: Microsoft DCOM (and ActiveX), CORBA, XML-RPC and SOAP (Web Services, XML is the IDL and HTTP is the network protocol)
RPC

- RPCs are embedded in the client portion of the application program
  - Not a standalone discreet middleware layer
- When the client code is compiled, a local “stub” is generated for the client, and when the application requires a remote function the stub is invoked to provide **synchronous** calls between the client and server
- An RPC is initiated by the caller (client) sending a request message to a remote system (the server) to execute a certain procedure using supplied arguments.
- A result message is returned to the caller.
RPC

CLIENT

Client functions

Client Stub

Network

SERVER

Server functions

Server Stub

Network

1.  
2.  
3.  
4.  
5.  
6.  
7.  
8.  
9.  
10.
RPC Issues

- How to pass parameters? (can’t pass by reference, and have to package complex structures)
- How to represent data? (different data sizes and representations)
- How to find the servers? (need to find the host and port)
- Which transport protocol?
- How to handle errors (servers disappearing, network problems)
- Semantics for calling the remote procedures?
- Performance? (extra steps to package data, calls stubs, network, …)
- Security?
RPC Variants

- RPC first discussed in 1976, first implementations for single processors in late 1970's

- **First Generation Implementations:**
  - One of the earliest examples for distributed systems is Sun RPC (early 80's, Open Network Computing architecture ONC RPC)
  - DCE RPC: Distributed Computing Environment RPC (designed by Open Software Foundation)
  - Sun/DCE RPC does not provide support for instantiating remote objects from remote classes, tracking instances of objects, or support for polymorphism

- **2nd Generation Object Based Implementations**
  - Microsoft DCOM (Distributed Object Component Model), object oriented implementation (1992 OLE “object linking and embedding”, evolved into COM “component object model”, DCOM introduced in 1996)
  - CORBA (Common Object Resource Broker Architecture) developed by industry consortium called the Object Management Group.
  - Java RMI (Remote Method Invocation)

- **3rd Generation Web Service Based Implementations**
  - XML-RPC, SOAP, Microsoft .NET
3. RPC and RMI Examples

- Many existing versions are not standardized and interoperable or are not suitable for scientific computing

- GridRPC
  - RPC model and API for grids, provides standard RPC semantics but also high level abstraction
  - Dynamic resource discovery and scheduling, security (GSI), fault tolerance.
  - Scientific IDL, server-side-only IDL management (simplify client-side stubs and state)
  - Prototypes: Ninf, NetSolve

- Java RMI
  - Object oriented, supports all java datatypes, garbage collection.
  - Program running in one JVM can invoke methods of other objects in different JVMs
4. Hybrid Programming Models

- Multiple models to enable running across e.g. a shared-address space (SMP) and Grid

- Examples:
  - OpenMP (multithreaded model) & MPI (message passing model) (requires threadsafe MPI)
  - OpenMP & RPC (e.g. OmniRPC)
  - Multithreading, RMI, Message passing (e.g. MPJ or Message Passing Java)
5. Peer-to-Peer Models

- Resources that traditionally would be clients are now act as both server and client
- Ian Taylor talked about P2P and the Grid in last lecture
5. Peer to Peer Examples

- **JXTA**
  - Open P2P protocols, defined as XML messages
  - Peers can form self organized and self configured groups with no centralized management
  - JXTA protocols advertise and discover resources, form and join subgroups, cooperate to route messages
6. Grid APIs Models

- Abstract high level application oriented interface to the Grid via API.
- Language independent specification, implementations in multiple languages.
- Independent of underlying programming model and implementation.
- Examples:
  - Grid Application Toolkit (GAT)
  - Simple API for Grid Applications (SAGA)
- We will be revisiting this.
7. Application Frameworks

- Entire application programming environments and toolkits with their own methods for grid/distributed computing

- Examples:
  - Cactus Computational Toolkit ([www.cactuscode.org](http://www.cactuscode.org))
  - Supports parallel I/O, checkpointing, computational steering etc in a Grid environment.
  - Enhancements to efficiently use MPICH-G2
  - Modules for Grid operations e.g. Spawning, migration, CGAT
  - Application developers with Cactus do not need to change their code to use the Grid!!
8. Component Models

- **Component**: Encapsulated part of a software system that implements some specific functionality or a set of capabilities.

- **A component model defines**:
  - Component properties
  - Exposed component interfaces
  - Infrastructure needed to support component interfaces (packing, deployment, runtime management)

- **Different to objects**
  - Multiple views per component
  - Extensibility (higher level of abstraction)
  - Higher level execution environment (components define a runtime execution environment)

- **Examples**: CORBA 3 Component Model, COM/DCOM, Enterprise Java Beans, CCA.
Web Services are a variant of RPC (with XML as the IDL and HTTP as the transport protocol).

Open Grid Services Architecture is a (still being defined) Grid architecture based on web services and technologies.

Services themselves are programming language and programming model neutral.

OGSA defines semantics of grid service instance: how is it created, names, lifetime determined, how to communicate with it.

GT4 is an OGSA implementation.
Required Reading

- *Grid Programming Models: Current Trends, Issues and Directions*
  - Craig Lee and Domenico Talia
Coursework 4

- For next Wednesday
  - Write a comparison of web services with RMI and RPC.
- Chirag ....