Ibis tutorial
&
hands-on session

Jason Maassen
jason@cs.vu.nl

Thursday 13 October 2005
Sophia Antipolis
Overview

- Philosophy / design / implementation
  - Why do we need Ibis?
  - Ibis design
  - Performance
  - Cool features
Overview

• Programming models
  • IPL (bare bones Ibis)
  • RMI (remote invocation)
  • GMI (group communication)
  • Satin (divide and conquer)
  • MPJ (MPI to Java binding)

• Hands-on session
  • How to roll your own Ibis applications
We are interested in...

• **Parallel** applications on “the Grid”
  • Single site runs
    • Grid == big collection of clusters
      • Only communicate within cluster
      • Use fast local network (Myrinet/Infiniband/…)

• **Multi site** runs
  • Grid == big processor pool
    • Communicate between clusters
    • Use regular network & internet
So why Ibis?

- Ideally, grid computing should be “fire and forget”
  - Develop application locally
  - Submit to some grid scheduler which
    - Finds some suitable site(s)
    - Transfers your application and data to the sites, and runs it.
    - Returns the result
Problems

• Lots of problems
  • Resource selection
  • Data transfer
  • Security and authentication
  • ...
  • Heterogeneity

• Globus, Gridlab (GAT), etc.
Problems

- Grids are heterogeneous:
  - Intel / PowerPC / Mips / Arm / ...
  - Windows / Linux / Unix / OSX / ...
  - Different OS/library/tool versions

- Compiled (C/MPI) apps. huge pain:
  - Need executable for every combination of CPU/network/OS/libraries etc.
  - Makes 'fire & forget' runs really hard...
Solution (partly)

- So, we use Java instead of C or Fortran
  - No recompilation required
  - Runs (almost) anywhere
    - Doesn't work on supercomputers (Hitachi SR8000, IBM BlueGene, etc.)
    - Most sites have clusters anyway
  - Acceptable performance

- But ... only part of the solution!
How about 'portable' communication?

- Class libraries are portable, but ...
  - Sockets are too low-level
  - RMI model/performance is limited

- Most parallel libraries not portable ...
  - MpiJava requires native code
    - recompilation
    - no malleable runs
Ibis

• Solution: Ibis!
  • A “run-anywhere” communication library
  • Just send it along with your application!

• Flexible communication models
  • More than just unicast communication
    • More about this later
  • Malleability & Fault-Tolerance
    • Change number of machines during the run
Ibis

- Portability vs. performance
  - On a single site run you often want to use the fast local network

- Ibis allows specialized implementation
  - Designed for Myrinet, Infiniband, etc.
  - Usually use native code
  - Installed in advance
    - not portable
    - cannot be shipped with application
Ibis

- As a result, there may be multiple Ibis available on a site
  - Automatically choose 'best' at startup
  - Based on requirements specified by
    - Application & user (using properties)

- Not every impl. needs all features
  - Pick one at startup that suits your needs....
Ibis Design

Ibis Portability Layer (IPL)
Ibis Design

Ibis Portability Layer (IPL)

TCP  UDP  P2P
Ibis Design

Ibis Portability Layer (IPL)

TCP  UDP  P2P  GM  Panda  MPI
Ibis Design

Ibis Portability Layer (IPL)

RMI  GMI  Satin  MPJ  RepMI

TCP  UDP  P2P  GM  Panda  MPI
Ibis Design

Ibis Portability Layer (IPL)
Ibis Design

Application

Ibis Portability Layer (IPL)

TCP  UDP  P2P  GM  Panda  MPI  RepMI  RMI  GMI  Satin  MPJ
Ibis Design

Application

ProActive

Ibis Portability Layer (IPL)

RMI

GMI

Satin

MPJ

RepMI

TCP

UDP

P2P

GM

Panda

MPI
Ibis Portability Layer

• Basic Ibis interface
  • Reasonably simple (5 classes & 12 interf.)

• Contains methods for
  • Loading an Ibis
  • Malleability (adding & removing machines)
  • Connection handling
  • Communication primitives (low-level)
IPL Communication

- 'Low-level' communication model
- Unidirectional pipes
- Two end points
- Connection oriented

![Diagram of send port and receive port connection]
Send & receive ports

- Can be connected in arbitrary ways
Send & receive ports

- Can be connected in arbitrary ways
- One to one (unicast) ...
Send & receive ports

- Can be connected in arbitrary ways
- ... one to many (multicast) ...
Send & receive ports

- Can be connected in arbitrary ways
- ... many to one ...
Send & receive ports

- Can be connected in arbitrary ways
- ... or some combination!
Send & receive ports

• Advantages:
  • Very simple & abstract model
  • Easy to implement using TCP/UDP/MPI/etc.
  • Allows multicast, many-to-one, etc.
    • Useful for parallel programs
  • Allows efficient implementation
    • Can be implemented using efficient low-level primitives (i.e., mpi-broadcast)
    • Other models do prevent this (e.g., RMI)
Send & receive ports

- Disadvantage:
  - Simplicity may cause some overhead...
  - Example: need two pairs for RPC / RMI
Port Types

- All ports have a type consisting of:
  - Unique name
  - Set of properties, e.g.:
    - Supports unicast and multicast
    - Is Reliable
    - Is fifo ordered
    - Supports object serialization
    - ...

Port Types

- Defined at runtime
  - Specify name and set of properties
- Types must match when connecting!
Port Types

- Defined at runtime
  - Specify name and set of properties

- Types must match when connecting!
Port Types

- Forces programmer to specify how each communication channel is used
  - Prevents bugs
    - Exception when contract is breached
  - Allows efficient impl. to be selected
    - Unicast only?
    - Bytes only?
    - Saves a lot of overhead!
Connection setup

- Need the ReceivePortIdentifier
  - Uniquely identifies a receiveport
- Created when ReceivePort is created
- May also have unique name (String)
  - Human-readable (usually)
  - Use for registry lookup
Connection setup

Create ReceivePort

“server”
Connection setup

Registry

register("server")
Connection setup

lookup("server")
Connection setup

Registry

ID
Connection setup

Registry

connect
Connection setup

- Advantage of ReceivePortIdentifiers
  - Hides implementation details
  - Independent of
    - IP-addresses
    - Host names
    - Port numbers
    - MPI-ranks
    - etc...

- Abstract way of addressing
Messages

- Ports communicate using 'messages'
- Contain read or write methods for
  - Primitive types (byte, int, ...)
  - Object
  - Arrays slices (partial write / read in place)
- Unlimited message size
Messages

• Get WriteMessage from SendPort
Messages

- Write data into WriteMessage
Messages

• Finish the WriteMessage
Messages

• Data is send to ReceivePort
Messages

- ReceivePort produces ReadMessage
  - Explicit receive or callback (upcall)
Messages

- Read data from ReadMessage
Messages

- Finish the ReadMessage
Messages

- Done!
Messages or streams?

- Message size is unlimited
  - Data may be forwarded at any time
  - Both S. & R. messages alive at same time
  - There's streaming!
Restrictions

- Must write and read data in same order
- A port can have only one message 'alive' at a time
- Not thread safe (but ports are)
- If there is no receiver, the sender may block (there may be flow-control)
Serialization

- Ibis supports 4 types of serialization
  - Bytes (i.e., no serialization at all)
  - Data (only primitive types/arrays)
  - Sun (standard Sun serialization)
  - Ibis (efficient Ibis serialization)

- To select one of the last two use 'Object'
Ibis Serialization

- Based on bytecode-rewriting
  - Adds serialization and deserialization code to serializable types
  - Prevents reflection overhead during (de-)serialization
  - Has fallback mechanism for non-rewritten classes
- Future work: runtime rewriting
Short Recap

- First create PortType
- PortType creates Send & ReceivePort
  - Type is checked when connecting
- Use ReceivePortID's to connect
  - Abstract addressing
- Use Messages to communicate
  - Allows streaming
  - 4 types of serialization
Creating an Ibis

• First step in application
  • IPL is only abstract classes & interfaces

• Ibis selects implementation for you
  • Multiple may be available
  • Selected on the basis of properties
    • Specify the needs of the application
Selecting an Ibis

Properties:
- closed world
- data ser.
- explicit rec.
- unicast

Ibis.createIbis(...)

vrije Universiteit
Selecting an Ibis

Ibis.createIbis()

Find all Ibis implementations

TCPIbis

.jar

application jar files

local disk

MPI-Ibis

Pandalbis

Properties:
- closed world
- data ser.
- explicit rec.

.jar files
Selecting an Ibis

Ibis.createIbis()

Select 'best' implementation

Properties:
- closed world
- data ser.
- explicit rec.
- unicast
- local disk
- .jar
- application jar files
- TCPIbis
- MPI-Ibis
- Pandalbis

Efficient Java-based grid computing
Selecting an Ibis

\[ \text{Ibis.createIbis}() \]

Return new instance

Properties:
- Closed world
- Data ser.
- Explicit rec.
Properties

• Usual Java properties
  • Set of key-value pairs
    • “serialization”, “object”
    • “communication”, “OneToOne”
    • “worldmodel”, “open”

• Very flexible
  • good and bad
Properties

• Good
  • Introduce features without IPL changes
    • Just add more properties
  • Allows impl. specific properties

• Bad
  • No compile time checks (only runtime)
    • Just strings
    • Sensitive to typos
## Example

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worldmodel</td>
<td>Open</td>
<td>Support malleability</td>
</tr>
<tr>
<td></td>
<td>Closed</td>
<td>Fixed set of machines</td>
</tr>
<tr>
<td>Serialization</td>
<td>Byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OneToOne</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OneToMany</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ManyToMany</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ExplicitReceive</td>
<td>Support explicit rec.</td>
</tr>
<tr>
<td></td>
<td>AutoMessageUpcalls</td>
<td>Automatic callback</td>
</tr>
<tr>
<td></td>
<td>PollingMessageUpcalls</td>
<td>Callback triggered by polling</td>
</tr>
<tr>
<td></td>
<td>ConnectionUpcalls</td>
<td>Callback when machines join or leave</td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Property</th>
<th>TCP</th>
<th>Ibis Panda</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worldmodel</td>
<td>Open</td>
<td>Closed</td>
<td>Closed</td>
</tr>
<tr>
<td>Serialization</td>
<td>B/D/O</td>
<td>B/D/O</td>
<td>B/D/O</td>
</tr>
<tr>
<td>Communication</td>
<td>OO/OM/MO</td>
<td>OO/OM/MO</td>
<td>OO/OM/MO</td>
</tr>
<tr>
<td>Explicit receive</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AutoMessageUpcalls</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>PollingMessageUpcalls</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ConnectionUpcalls</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ConnectionDowncalls</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Malleability

- Ibis can notify the application if a machine joins or leaves
  - Callbacks/upcalls to a 'ResizeHandler'
  - Calls are delivered in the same order on all machines

- Each 'Ibis' has a unique IbisIdentifier
  - Abstract identification of the machine
  - Impl. using IP-addresses / MPI-ranks / etc.
Elections

- Ibis offers an 'election' mechanism
  - Allows a group of machines to determine who's in charge

- Each election
  - Has a name (String)
  - Produces an IbisIdentifier identifying the winner
  - Is not democratic
Other cool features

- TCP Ibis supports
  - Parallel streams
    - For high-latency & high-bandwidth links
- NAT / firewall traversal
  - TCP splicing
  - Routing messages through external point
import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

        // Step 1: create ibis
        StaticProperties pl = new StaticProperties();
        pl.add("communication", "OneToOne, Reliable, ExplicitReceipt");
        pl.add("serialization", "object");
        Ibis ibis = Ibis.createIbis(pl, null);

        // Step 2: create porttype
        PortType type = ibis.createPortType("Test Type", pl);

        // Step 3: elect server
        Registry reg = ibis.registry();
        IbisIdentifier server = reg.elect("Server");
        boolean amServer = server.equals(ibis.identifier());

        if (amServer) {
            // Step 4, create port
            ReceivePort rp = type.createReceivePort("server");
            rp.enableConnections();

            // Step 5, receive message and read data
            ReadMessage rm = rp.receive();
            String tmp = (String) rm.readObject();
            rm.finish();
            System.out.println("Client says: " + tmp);

            // Step 6, close port
            rp.close();
        } else {
            // Step 4, create port, find receivePort and connect
            SendPort sp = type.createSendPort();
            sp.connect(reg.lookupReceivePort("server"));

            // Step 5, get message and write data
            WriteMessage wm = sp.newMessage();
            wm.writeObject("Hello World");
            wm.finish();

            // Step 6, close port
            sp.close();
        }

        ibis.end();
    }
}
Code example

```java
import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

        // Step 1: create ibis
        StaticProperties p1 = new StaticProperties();
        p1.add("communication", "OneToOne, Reliable, ExplicitReceipt");
        p1.add("serialization", "object");
        Ibis ibis = Ibis.createIbis(p1, null);

        // Step 2: create porttype
        PortType type = ibis.createPortType("Test Type", p1);

        // Step 3: elect server
        Registry reg = ibis.registry();
        IbisIdentifier server = reg.elect("Server");
        boolean amServer = server.equals(ibis.identifier());

        if (amServer) {
            // Step 4, create port
            ReceivePort rp = type.createReceivePort("server");
            rp.enableConnections();

            // Step 5, receive message and read data
            ReadMessage rm = rp.receive();
            String tmp = (String) rm.readObject();
            rm.finish();

            System.out.println("Client says: " + tmp);

            // Step 6, close port
            rp.close();
        } else {
            // Step 4, create port, find receivePort and connect
            SendPort sp = type.createSendPort();
            sp.connect(reg.lookupReceivePort("server");

            // Step 5, get message and write data
            WriteMessage wm = sp.newMessage();
            wm.writeObject("Hello World");
            wm.finish();

            // Step 6, close port
            sp.close();
        }

        ibis.end();
    }
}
```
import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

        // Step 1: create ibis
        StaticProperties p1 = new StaticProperties();
        p1.add("communication", "OneToOne, Reliable, ExplicitReceipt");
        p1.add("serialization", "object");

        Ibis ibis = Ibis.createIbis(p1, null);

        // Step 2: create porttype
        PortType type = ibis.createPortType("Test Type", p1);

        // Step 3: elect server
        Registry reg = ibis.registry();
        IbisIdentifier server = reg.elect("Server");
        boolean amServer = server.equals(ibis.identifier());

        if (amServer) {
            // Step 4, create port
            ReceivePort rp = type.createReceivePort("server");
            rp.enableConnections();

            // Step 5, receive message and read data
            ReadMessage rm = rp.receive();
            String tmp = (String) rm.readObject();
            rm.finish();
            System.out.println("Client says: "+ tmp);

            // Step 6, close port
            rp.close();
        } else {
            // Step 4, create port, find receivePort and connect
            SendPort sp = type.createSendPort();
            sp.connect(reg.lookupReceivePort("server"));

            // Step 5, get message and write data
            WriteMessage wm = sp.newMessage();
            wm.writeObject("Hello World");
            wm.finish();

            // Step 6, close port
            sp.close();
        }

        ibis.end();
    }
}
import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

        // Step 1: create ibis
        StaticProperties pl = new StaticProperties();
        pl.add("communication", "OneToOne, Reliable, ExplicitReceipt");
        pl.add("serialization", "object");

        Ibis ibis = Ibis.createIbis(pl, null);

        // Step 2: create porttype
        PortType type = ibis.createPortType("Test Type", pl);

        // Step 3: elect server
        Registry reg = ibis.registry();
        IbisIdentifier server = reg.elect("Server");
        boolean amServer = server.equals(ibis.identifier());

        if (amServer) {
            // Step 4, create port
            ReceivePort rp = type.createReceivePort("server");
            rp.enableConnections();

            // Step 5, receive message and read data
            ReadMessage rm = rp.receive();
            String tmp = (String) rm.readObject();
            rm.finish();

            System.out.println("Client says: " + tmp);

            // Step 6, close port
            rp.close();
        } else {

            // Step 4, create port, find receivePort and connect
            SendPort sp = type.createSendPort();
            sp.connect(reg.lookupReceivePort("server"));

            // Step 5, get message and write data
            WriteMessage wm = sp.newMessage();
            wm.writeObject("Hello World");
            wm.finish();

            // Step 6, close port
            sp.close();
        }

        ibis.end();
    }
}
import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

        // Step 1: create ibis
        StaticProperties pl = new StaticProperties();
        pl.add("communication", "OneToOne, Reliable, ExplicitReceipt");
        pl.add("serialization", "object");
        Ibis ibis = Ibis.createIbis(pl, null);

        // Step 2: create porttype
        PortType type = ibis.createPortType("Test Type", pl);

        // Step 3: elect server
        Registry reg = ibis.registry();
        IbisIdentifier server = reg.elect("Server");
        boolean amServer = server.equals(ibis.identifier());

        if (amServer) {
            // Step 4, create port
            ReceivePort rp = type.createReceivePort("server");
            rp.enableConnections();

            // Step 5, receive message and read data
            ReadMessage rm = rp.receive();
            String tmp = (String) rm.readObject();
            rm.finish();

            System.out.println("Client says: "+ tmp);
        } else {
            // Step 4, create port, find receivePort and connect
            SendPort sp = type.createSendPort();
            sp.connect(reg.lookupReceivePort("server"));

            // Step 5, get message and write data
            WriteMessage wm = sp.newMessage();
            wm.writeObject("Hello World");
            wm.finish();

            // Step 6, close port
            sp.close();
        }

        ibis.end();
    }
}
import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

        // Step 1: create ibis
        StaticProperties pl = new StaticProperties();
        pl.add("communication", "OneToOne, Reliable, ExplicitReceipt");
        pl.add("serialization", "object");
        Ibis ibis = Ibis.createIbis(pl, null);

        // Step 2: create porttype
        PortType type = ibis.createPortType("Test Type", pl);

        // Step 3: elect server
        Registry reg = ibis.registry();
        IbisIdentifier server = reg.elect("Server");
        boolean amServer = server.equals(ibis.identifier());

        if (amServer) {

            // Step 4, create port, find receivePort and connect
            ReceivePort rp = type.createReceivePort("server");
            rp.enableConnections();

            // Step 5, receive message and read data
            ReadMessage rm = rp.receive();
            String tmp = (String) rm.readObject();
            rm.finish();

            System.out.println("Client says: " + tmp);

            // Step 6, close port
            rp.close();
        } else {

            // Step 4, create port, find receivePort and connect
            SendPort sp = type.createSendPort();
            sp.connect(reg.lookupReceivePort("server"));

            // Step 5, get message and write data
            WriteMessage wm = sp.newMessage();
            wm.writeObject("Hello World");
            wm.finish();

            // Step 6, close port
            sp.close();
        }

        ibis.end();
    }
}
import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

        // Step 1: create ibis
        StaticProperties pl = new StaticProperties();
        pl.add("communication", "OneToOne, Reliable, ExplicitReceipt");
        pl.add("serialization", "object");
        Ibis ibis = Ibis.createIbis(pl, null);

        // Step 2: create porttype
        PortType type = ibis.createPortType("Test Type", pl);

        // Step 3: elect server
        Registry reg = ibis.registry();
        IbisIdentifier server = reg.elect("Server");
        boolean amServer = server.equals(ibis.identifier());

        if (amServer) {
            // Step 4, create port
            ReceivePort rp = type.createReceivePort("server");
            rp.enableConnections();

            // Step 5, receive message and read data
            ReadMessage rm = rp.receive();
            String tmp = (String) rm.readObject();
            rm.finish();

            System.out.println("Client says: "+ tmp);

            // Step 6, close port
            rp.close();
        } else {

            // Step 4, create port, find receivePort and connect
            SendPort sp = type.createSendPort();
            sp.connect(reg.lookupReceivePort("server"));

            // Step 5, get message and write data
            WriteMessage wm = sp.newMessage();
            wm.writeObject("Hello World");
            wm.finish();

            // Step 6, close port
            sp.close();

        }

    }

    ibis.end();
}

// Step 5, get message and write data
WriteMessage wm = sp.newMessage();
wm.writeObject("Hello World");
wm.finish();

}
import ibis.ipl.*;

public class Example {
    public static void main(String args[]) throws Exception {
        // Step 1: create ibis
        StaticProperties pl = new StaticProperties();
        pl.add("communication", "OneToOne, Reliable, ExplicitReceipt");
        pl.add("serialization", "object");
        Ibis ibis = Ibis.createIbis(pl, null);

        // Step 2: create porttype
        PortType type = ibis.createPortType("Test Type", pl);

        // Step 3: elect server
        Registry reg = ibis.registry();
        IbisIdentifier server = reg.elect("Server");
        boolean amServer = server.equals(ibis.identifier());

        if (amServer) {
            // Step 4, create port
            ReceivePort rp = type.createReceivePort("server");
            rp.enableConnections();

            // Step 5, receive message and read data
            ReadMessage rm = rp.receive();
            String tmp = (String) rm.readObject();
            rm.finish();
            System.out.println("Client says: " + tmp);

            // Step 6, close port
            rp.close();
        } else {
            // Step 4, create port, find receivePort and connect
            SendPort sp = type.createSendPort();
            sp.connect(reg.lookupReceivePort("server"));

            // Step 5, get message and write data
            WriteMessage wm = sp.newMessage();
            wm.writeObject("Hello World");
            wm.finish();

            // Step 6, close port
            sp.close();
        }
    }
}
import ibis.ipl.*;

public class Example {
    public static void main(String args[]) throws Exception {

        // Step 1: create ibis
        StaticProperties pl = new StaticProperties();
        pl.add("communication", "OneToOne, Reliable, ExplicitReceipt");
        pl.add("serialization", "object");

        Ibis ibis = Ibis.createIbis(pl, null);

        // Step 2: create porttype
        PortType type = ibis.createPortType("Test Type", pl);

        // Step 3: elect server
        Registry reg = ibis.registry();
        IbisIdentifier server = reg.elect("Server");
        boolean amServer = server.equals(ibis.identifier());

        if (amServer) {
            // Step 4, create port
            ReceivePort rp = type.createReceivePort("server");
            rp.enableConnections();

            // Step 5, receive message and read data
            ReadMessage rm = rp.receive();
            String tmp = (String) rm.readObject();
            rm.finish();

            System.out.println("Client says: " + tmp);

            // Step 6, close port
            rp.close();
        } else {

            // Step 4, create port, find receivePort and connect
            SendPort sp = type.createSendPort();
            sp.connect(reg.lookupReceivePort("server"));

            // Step 5, get message and write data
            WriteMessage wm = sp.newMessage();
            wm.writeObject("Hello World");
            wm.finish();

            // Step 6, close port
            sp.close();
        }
    }
}

Code example

```java
import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

        // Step 1: create ibis
        StaticProperties pl = new StaticProperties();
        pl.add("communication", "OneToOne, Reliable, ExplicitReceipt");
        pl.add("serialization", "object");

        Ibis ibis = Ibis.createIbis(pl, null);

        // Step 2: create porttype
        PortType type = ibis.createPortType("Test Type", pl);

        // Step 3: elect server
        Registry reg = ibis.registry();
        IbisIdentifier server = reg.elect("Server");
        boolean amServer = server.equals(ibis.identifier());

        if (amServer) {
            // Step 4, create port
            ReceivePort rp = type.createReceivePort("server");
            rp.enableConnections();
            // Step 5, receive message and read data
            ReadMessage rm = rp.receive();
            String tmp = (String) rm.readObject();
            rm.finish();
            System.out.println("Client says: " + tmp);
            // Step 6, close port
            rp.close();
        } else {
            // Step 4, create port, find receivePort and connect
            SendPort sp = type.createSendPort();
            sp.connect(reg.lookupReceivePort("server"));
            // Step 5, get message and write data
            WriteMessage wm = sp.newMessage();
            wm.writeObject("Hello World");
            wm.finish();
            // Step 6, close port
            sp.close();
        }
    }
}
```
Code Example

• Live demo
Nameserver

- Used by Ibises to find each other
  - Needs to be in a well known place
  - Used to implements joins & leaves
  - Used to implement Registry
  - Supports multiple namespaces
    - So 'conflicting' apps can use the same server
Higher level programming models

- Remote Method Invocation (RMI)
- Group Method Invocation (GMI)
- Satin (Divide & Conquer)
- MPJ (MPI Java 'standard')
- Others are being developed
  - Balutek (data parallel)
  - Replicated Method Invocation (RepMI)
Satin

- Parallel Divide-and-conquer
  - Divide work into independent parts
  - Spawn sub-jobs
  - Combine sub-results
  - Repeat recursively
- Master-Worker is a subset of this
  - Only one level of recursion
- Targeted at the grid (and clusters)
public long fib(int n) {
    if (n < 2) return n;

    long x = fib(n - 1);
    long y = fib(n - 2);

    return x + y;
}
Parallel Fibonacci

```java
interface FibInterface extends ibis.satin.Spawnable {
    public long fib(int n);
}

public long fib(int n) {
    if (n < 2) return n;
    long x = fib(n - 1);
    long y = fib(n - 2);
    sync();
    return x + y;
}
```
Parallel Fibonacci

interface FibInterface extends ibis.satin.Spawnable {
    public long fib(int n);
}

public long fib(int n) {
    if (n < 2) return n;
    long x = fib(n - 1);
    long y = fib(n - 2);
    sync();
    return x + y;
}
Parallel Fibonacci

```java
interface FibInterface extends ibis.satin.Spawnable {
    public long fib(int n);
}

public long fib(int n) {
    if (n < 2) return n;
    long x = fib(n - 1);
    long y = fib(n - 2);
    sync();
    return x + y;
}
```

Mark methods as Spawnable. They are allowed to run in parallel.

Wait until spawned methods are done.
Satin features

- Satin distributes jobs across machines
- Load-balancing is done automatically
  - Algorithm has been proven to be optimal on homogeneous systems
  - Additional highly-efficient grid-aware algorithms
Satin features

- Malleability
  - Add/remove machines on the fly

- Fault-tolerance
  - When a machine leaves suddenly (crashes) the others continue the computation and automatically recompute the lost work

- Shared Objects (added recently)
  - Allows machines to share 'global data'
Satin Applications

• More interesting applications
  • Numerical functions
  • N-body simulations
  • Game-tree search
  • Raytracer
  • Satisfiability solver
  • Grammar-based text analysis
  • Bioinformatics applications
  • ...

Ibis RMI

- Replacement for Sun RMI
  - Has the same interface
  - Used different stub compiler (rmic)
    - Generates Ibis specific stubs/skeletons
Ibis RMI

- Replacement for Sun RMI
  - Has the same interface
  - Used different stub compiler (rmic)
    - Generates Ibis specific stubs/skeletons
Ibis RMI

- Not interoperable with Sun RMI
  - uses a different protocol

- No socket factories
  - Ibis doesn't have to use sockets!

- No activatable objects
GMI

• Generalized RMI model
  • Allows communication with groups
    • A single stub refers to an entire group
  • Allows more 'advanced' communication
    • By offering different ways of forwarding a method invocation and handling the reply
GMI

- Group
  - Contains 1 or more objects
    - Fixed size (set when it is created)
  - All objects must implement the same group interface
    - But objects may have different type!
  - Unique name
    - Used in lookup (produces group reference)
  - Group members have rank
    - Ranks are 'per-group'
GMI Example
GMI Implementation
GMI Implementation
Group operations

• The group reference can be configured
  • How is a method invocation handled
  • How is the method result handled
  • Configuration per method

• Implemented by selecting different communication code in the generated stubs and skeletons
Invocation Schemes

- **Single**
  - Forward to 1 object in group

- **Group**
  - Forward to all objects in group

- **Personalized**
  - Forward to all objects, but personalize parameters for each target

- **Combined**
  - Combine several invocation into one, then forward to the group using one of the above
Single

foo()
Single

foo()
Group

foo()
Group
Personalized

$\text{foo([1,2,3])}$
Personalized
Combined

foo([1])

foo([2])
Combined

foo([1,2])
foo([2])
foo([1])
foo([1,2])
Combined
Reply handling schemes

- **Discard**
- **Return**
- **Forward**
  - Reply is forwarded to a separate object
- **Combine**
  - Multiple replies are combined into one
- **Personalize**
  - A personalized result is returned to each participant of a combined invocation
# GMI Communication

<table>
<thead>
<tr>
<th>Operation</th>
<th>Invocation</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMI</td>
<td>Single</td>
<td>Return</td>
</tr>
<tr>
<td>Async. RMI</td>
<td>Single</td>
<td>Discard</td>
</tr>
<tr>
<td>Future</td>
<td>Single</td>
<td>Forward</td>
</tr>
<tr>
<td>Broadcast</td>
<td>Group</td>
<td>Discard</td>
</tr>
<tr>
<td>Scatter</td>
<td>Personalized</td>
<td>Discard</td>
</tr>
<tr>
<td>Reduce results</td>
<td>Group</td>
<td>Combine (binomial)</td>
</tr>
<tr>
<td>Gather results</td>
<td>Group</td>
<td>Combine (flat)</td>
</tr>
<tr>
<td>Reduce inv.</td>
<td>Combine + Single</td>
<td>Discard</td>
</tr>
<tr>
<td>Gather inv.</td>
<td>Combine + Single</td>
<td>Discard</td>
</tr>
</tbody>
</table>
Code example

```java
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

class SimpleGroup extends GroupMember implements i_SimpleGroup {
    public SimpleGroup() {
        super();
    }
    public void ping() {
        System.out.println("ping");
    }
}
class MulticastNoReply {
    public static void main(String[] args) throws Exception {
        int rank = Group.rank();
        int size = Group.size();
        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }
        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);
        if (rank == 0) {
            // Perform lookup to get group reference
            i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");
            // Configure reference to perform group invocation
            GroupMethod m = Group.findMethod(g, "void ping");
            m.configure(new GroupInvocation(), new DiscardReply());
            // Perform the invocation
            g.ping();
        }
        // Done
        Group.exit();
    }
}
```
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {
    public SimpleGroup() {
        super();
    }
    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {
    public static void main(String[] args) throws Exception {
        int rank = Group.rank();
        int size = Group.size();
        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }
        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);
        if (rank == 0) {
            // Perform lookup to get group reference
            i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");
            // Configure reference to perform group invocation
            GroupMethod m = Group.findMethod(g,"void ping()");
            m.configure(new GroupInvocation(), new DiscardReply());
            // Perform the invocation
            g.ping();
        }
        // Done
        Group.exit();
    }
}
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {
    public SimpleGroup() {
        super();
    }
    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {
    public static void main(String[] args) throws Exception {
        int rank = Group.rank();
        int size = Group.size();
        Group.create("GroupNoReply", i_SimpleGroup.class, size);
        Gatherer g = new Gatherer();
        g.register("GroupNoReply");
        g.send(new InvCall(new MethodRef("ping"), new Object[0]));
    }
}

// Done
Group.exit();
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {
    public SimpleGroup() {
        super();
    }
    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {
    public static void main(String[] args) throws Exception {
        int rank = Group.rank();
        int size = Group.size();

        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }
        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);
        if (rank == 0) {
            // Perform lookup to get group reference
            i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");
            // Configure reference to perform group invocation
            GroupMethod m = Group.findMethod(g, "void ping()");
            m.configure(new GroupInvocation(), new DiscardReply());
            // Perform the invocation
            g.ping();
        }
        // Done
        Group.exit();
    }
}
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {
    public SimpleGroup() {
        super();
    }
    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {
    public static void main(String[] args) throws Exception {
        int rank = Group.rank();
        int size = Group.size();

        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }

        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);

        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }

        // Done
        Group.exit();
    }
}
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

class SimpleGroup extends GroupMember implements i_SimpleGroup {
    public SimpleGroup() {
        super();
    }
    public void ping() {
        System.out.println("ping");
    }
}

class MulticastNoReply {
    public static void main(String[] args) throws Exception {
        int rank = Group.rank();
        int size = Group.size();
        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }
        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);
        if (rank == 0) {
            // Perform lookup to get group reference
            i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");
            // Configure reference to perform group invocation
            GroupMethod m = Group.findMethod(g, "void ping()");
            m.configure(new GroupInvocation(), new DiscardReply());
            // Perform the invocation
            g.ping();
        }
        // Done
        Group.exit();
    }
}
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {

    public SimpleGroup() {
        super();
    }

    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {

    public static void main(String[] args) throws Exception {
        int rank = Group.rank();
        int size = Group.size();

        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }

        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);

        if (rank == 0) {
            // Perform lookup to get group reference
            i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");

            // Configure reference to perform group invocation
            GroupMethod m = Group.findMethod(g,"void ping()");  
            m.configure(new GroupInvocation(), new DiscardReply());

            // Perform the invocation
            g.ping();
        }

        // Done
        Group.exit();
    }
}
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {
    public SimpleGroup() {
        super();
    }

    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {
    public static void main(String[] args) throws Exception {
        int rank = Group.rank();
        int size = Group.size();

        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }

        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);

        if (rank == 0) {
            // Perform lookup to get group reference
            i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");

            // Configure reference to perform group invocation
            GroupMethod m = Group.findMethod(g,"void ping()");
            m.configure(new GroupInvocation(), new DiscardReply());

            // Perform the group invocation
            g.ping();
        }

        // Done
        Group.exit();
    }
}
Code Example

- Live demo
Function Objects

- Some operations need user defined functions
  - Personalizing a method invocation
  - Combining a result or invocation
  - Forwarding of results

- GMI uses function objects
  - Extend a class from the GMI package
Result Combiners

- Use 'combiner' to merge the results of an invocation
- FlatCombiner
  - Combines all results in one go
  - Similar to 'gather' operation of MPI
- BinomialCombiner
  - Pairwise combines results
  - Similar to 'reduce' operation of MPI
public class FlatCombiner {

    public boolean combine(boolean[] results, Exception[] ex)
    public byte combine(byte[] results, Exception[] ex)
    public char combine(char[] results, Exception[] ex)
    public short combine(short[] results, Exception[] ex)
    public int combine(int[] results, Exception[] ex)
    public long combine(long[] results, Exception[] ex)
    public float combine(float[] results, Exception[] ex)
    public double combine(double[] results, Exception[] ex)

    public Object combine(Object[] results, Exception[] ex)

    public void combine(Exception[] exceptions)
}
public class MyCombiner extends FlatCombiner {

    public int combine(int[] results, Exception[] ex) {

        int sum = 0;

        for (int i=0;i<results.length;i++) {
            sum += results[i];
        }

        return sum;
    }
}

FlatCombiner
```java
// Get a group reference
X g = (X) Group.lookup("your group");

// Configure reference to perform group invocation, 
// and combine the replies using 'MyCombiner'
GroupMethod m = Group.findMethod(g, "int get()");
m.configure(new GroupInvocation(),
            new CombineReply(new MyCombiner()));

// Perform the invocation
int result = g.get();
```
FlatCombiner

get()
FlatCombiner
FlatCombiner
FlatCombiner

MyCombiner

combine([42,64,8], [])
FlatCombiner

MyCombiner

114
FlatCombiner
FlatCombiner Demo

- Live demo
After the Break

• Hands-on session
  • Installing ibis
  • Running applications
    • Scripts and command line
  • Writing your own applications