The Distributed Application Debugger (DAD)

Michael Q. Jones & Matt B. Pedersen
University of Nevada Las Vegas

Introduction

- The Distributed Application Debugger is a debugging tool for parallel programs
- Targets the MPI platform
- Runs remotley even on private networks
- Has record and replay features.
- Integrates GDB

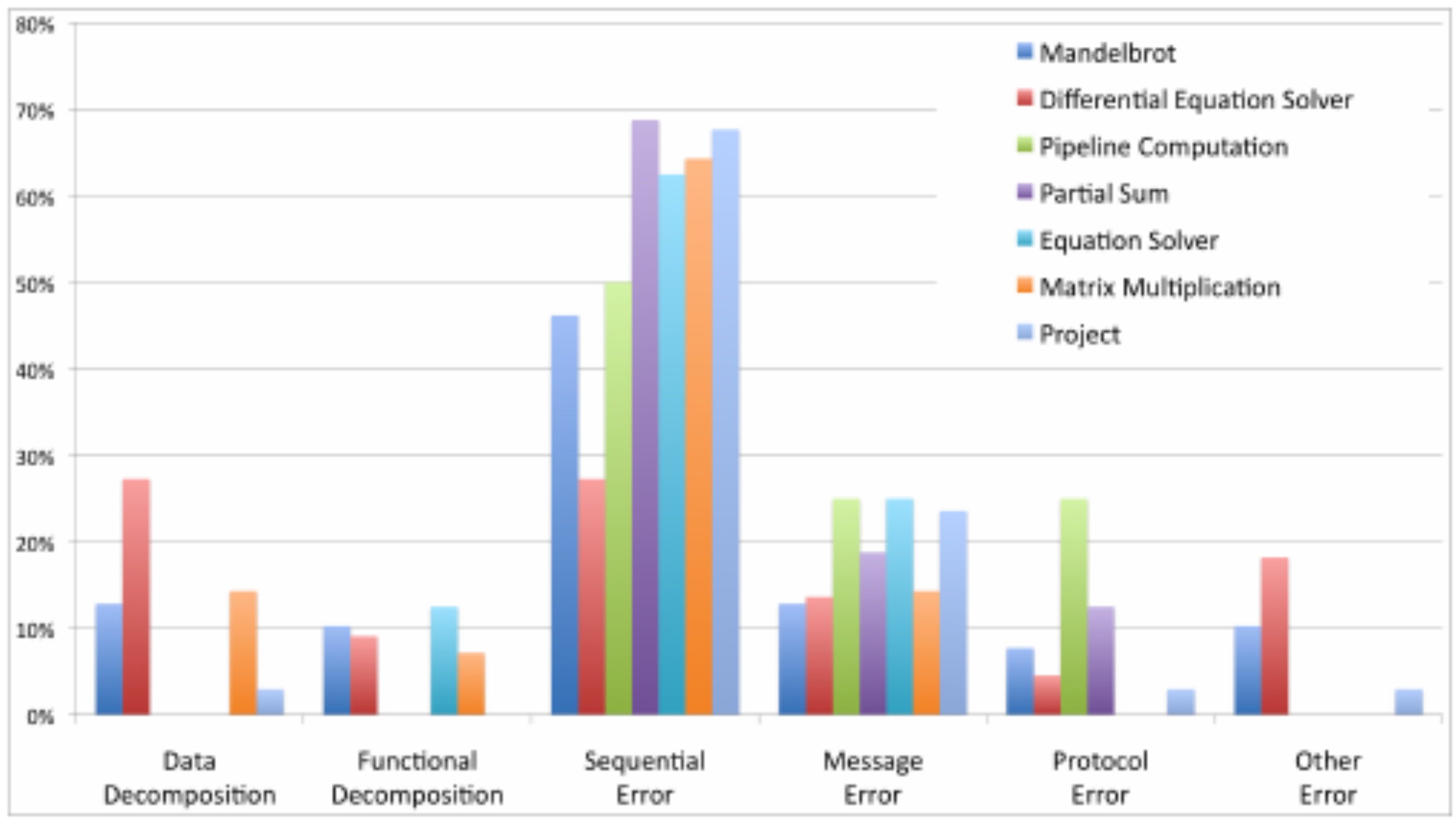
Why a new tool?

- Results from survey of students learning parallel programming concluded 3 things:
  - 1. Sequential errors are still frequent
  - 2. Message errors are time consuming
  - 3. Print statements are still used for debugging

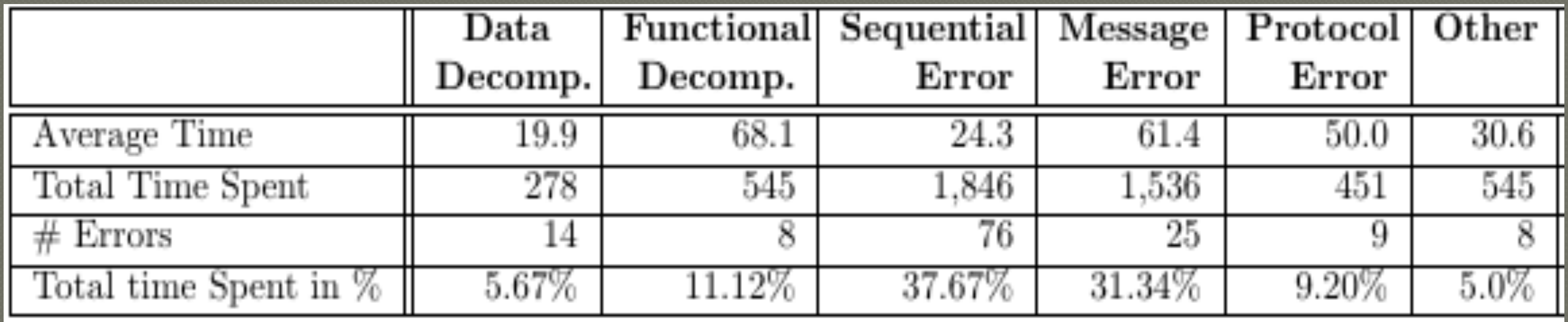
Survey

- Survey results categorized according to the domains of multilevel debugging
  - Sequential errors
  - Message errors
  - Protocol errors
- In addition to
  - Data decomposition errors
  - Functional decomposition errors

Survey Results



Survey Results

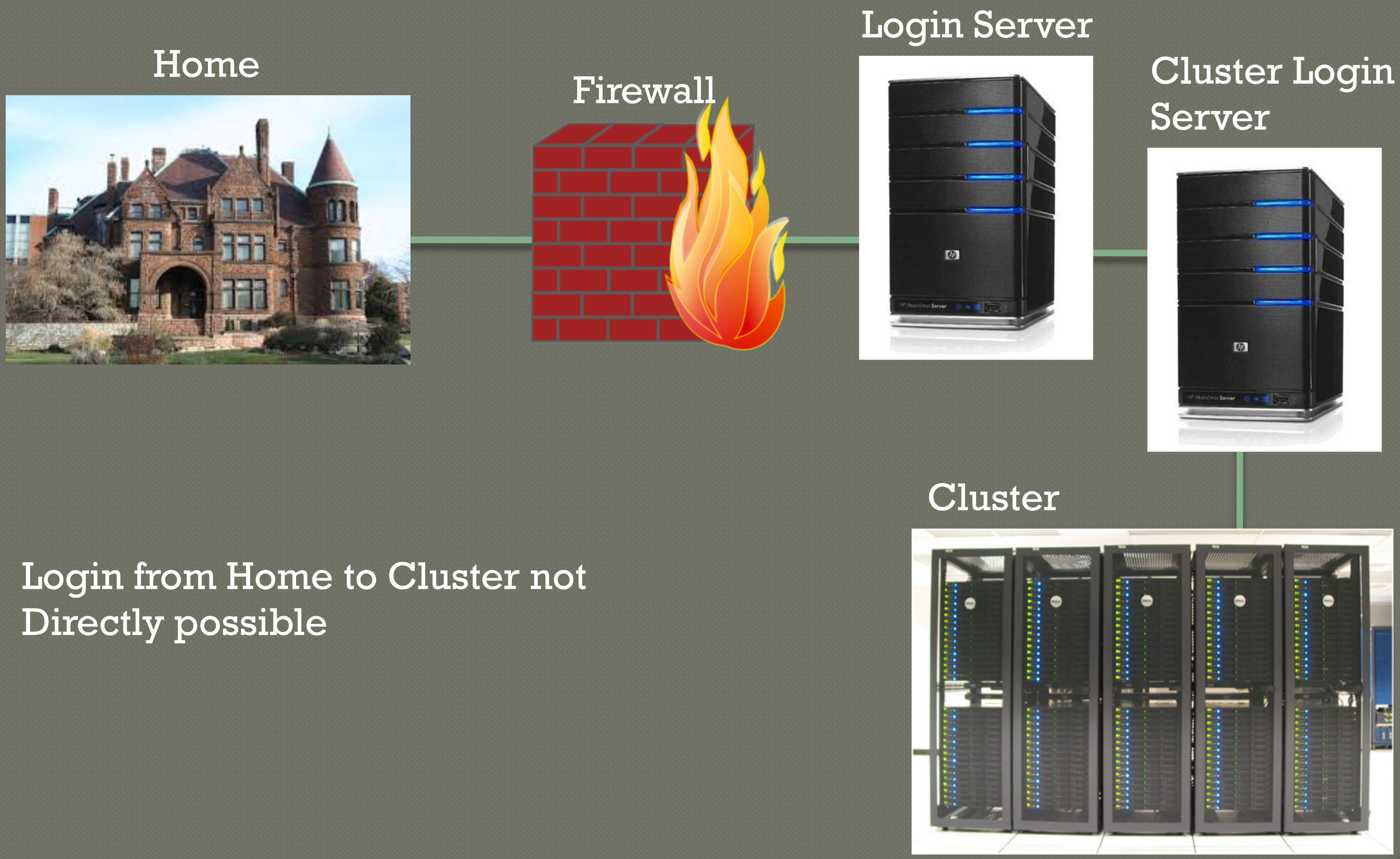


	Data Decomp.	Functional Decomp.	Sequential Error	Message Error	Protocol Error	Other
Average Time	19.9	68.1	24.3	61.4	50.0	30.6
Total Time Spent	278	545	1,846	1,536	451	545
# Errors	14	8	76	25	9	8
Total time Spent in %	5.67%	11.12%	37.67%	31.34%	9.20%	5.0%

The Components

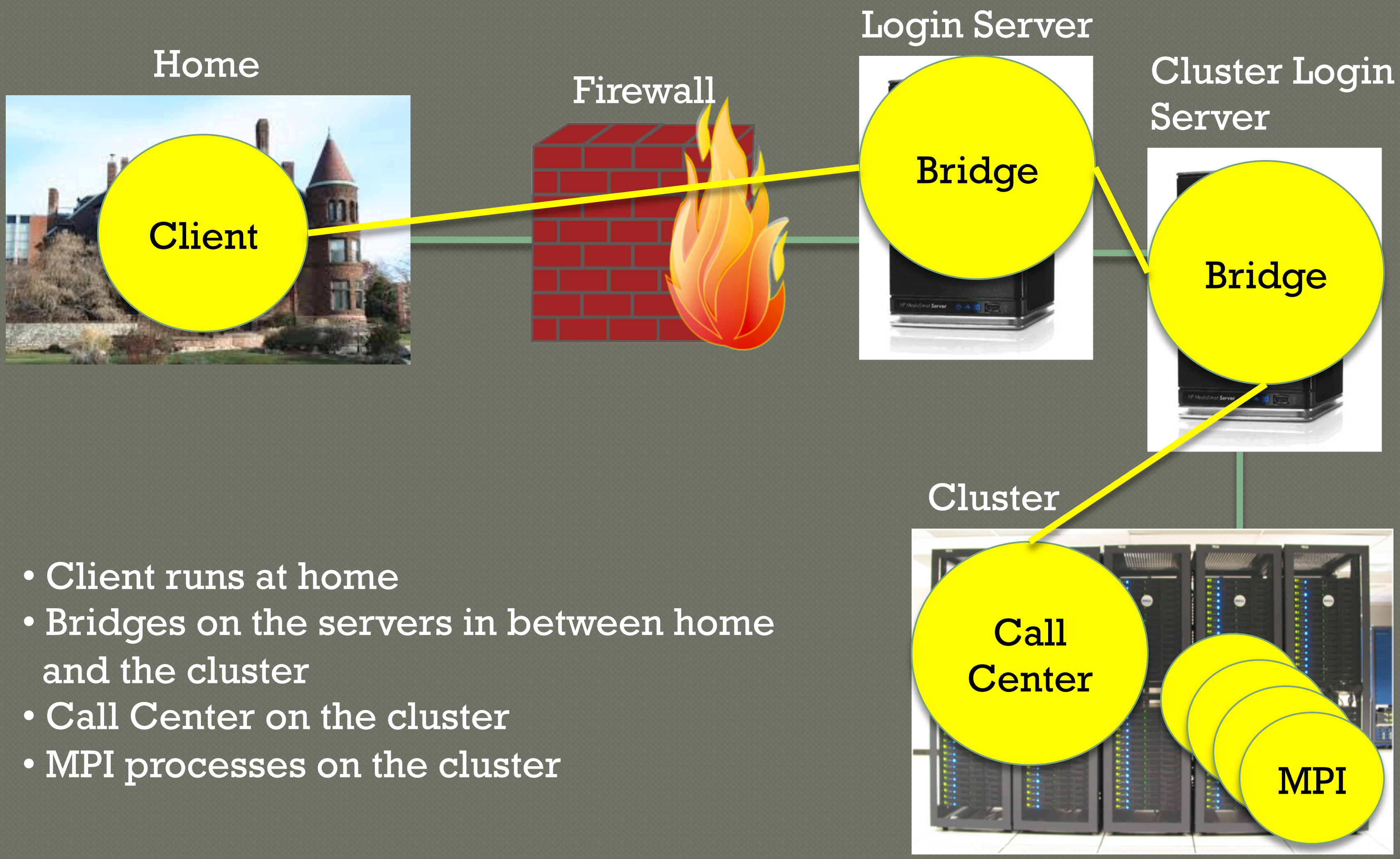
- The Client
  - The GUI interacting with the programmer
- The Call Center
  - A central messaging hub (running on the cluster) for
    - Routing messages from the MPI program to The Client
    - Routing commands from The Client to the MPI program
- Bridges
  - A relay application for passing data between The Client and The Call Center, when The Call Center is not directly accessible (cluster behind firewall)
- The Runtime
  - A libraries with wrapper code for the MPI functions (talks to The Call Center)

The Setup



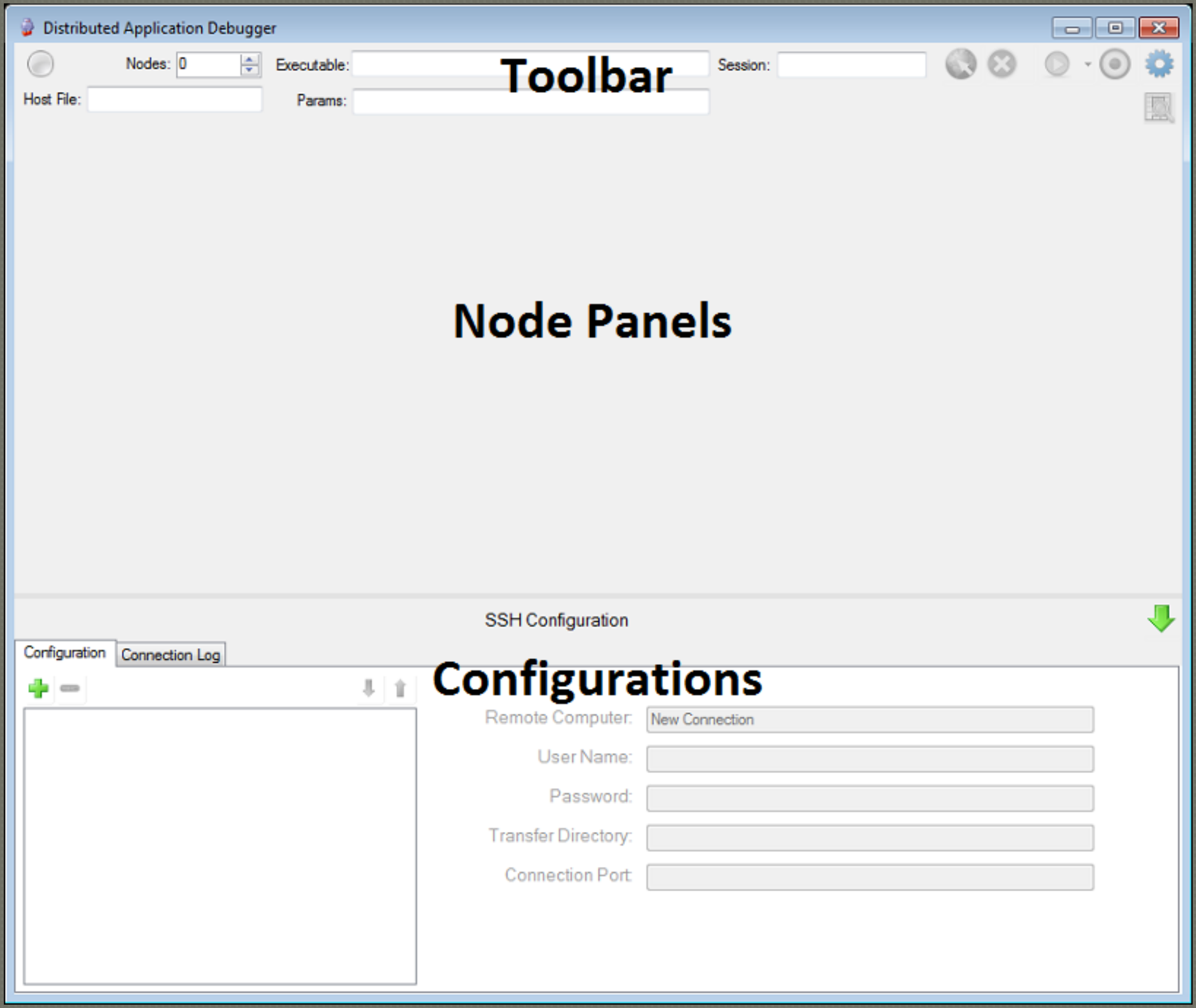
Login from Home to Cluster not Directly possible

The Setup



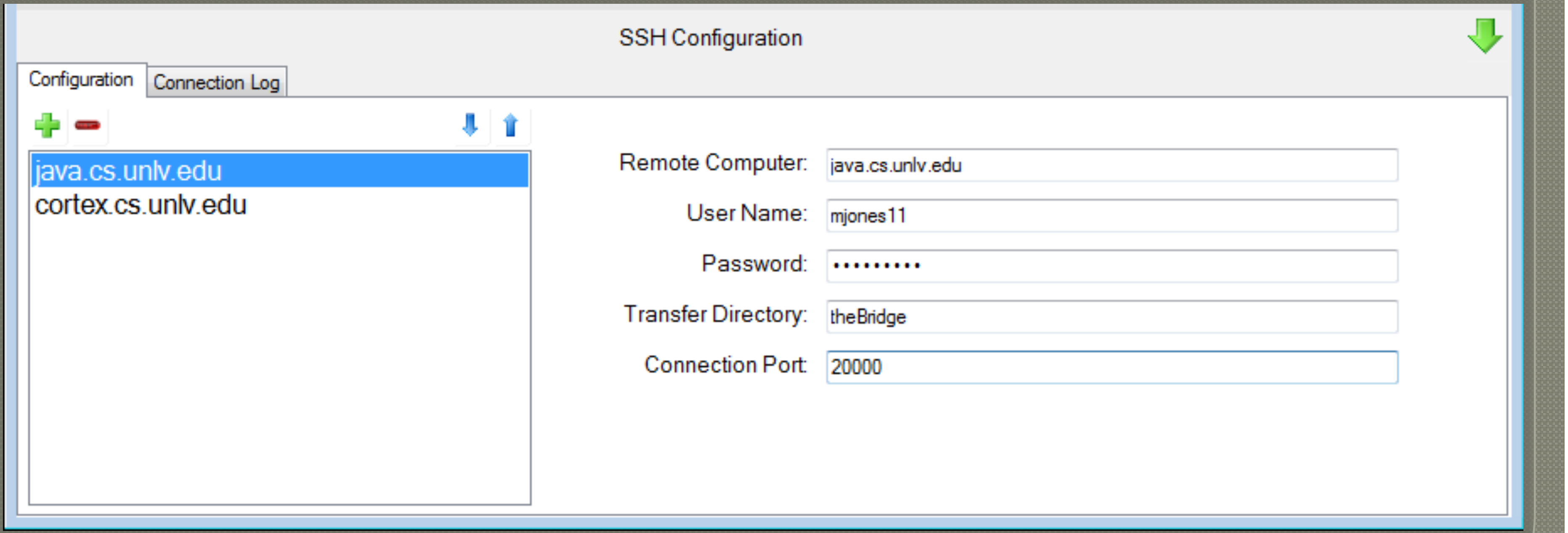
- Client runs at home
- Bridges on the servers in between home and the cluster
- Call Center on the cluster
- MPI processes on the cluster

The Distributed Application Debugger



Client Side

- The user provides a connection path and credentials on all machines



SSH Configuration

java.cs.unlv.edu
cortex.cs.unlv.edu

Remote Computer: java.cs.unlv.edu

User Name: mjones11

Password: ••••••••••

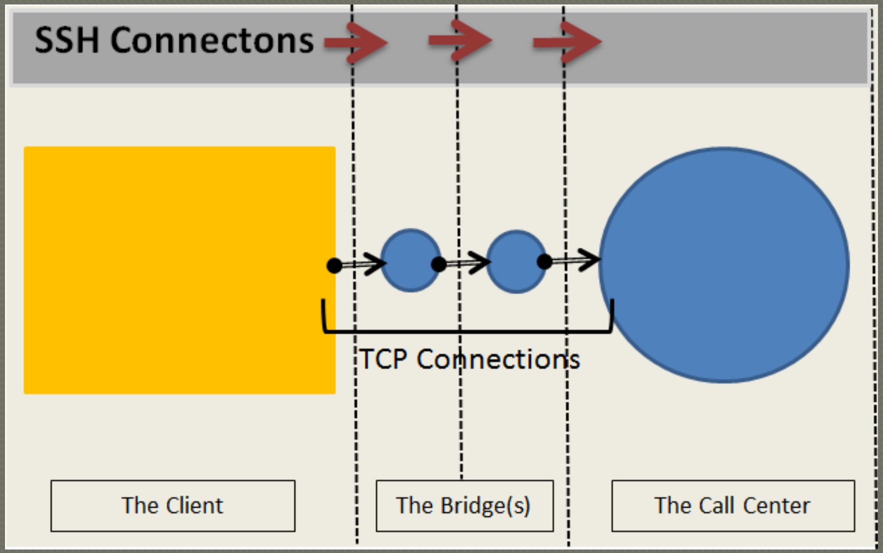
Transfer Directory: theBridge

Connection Port: 20000

Client Side

- The user provides a connection path and credentials on all machines
- The system initiates SSH connections to each configured computer and launches a Bridge or The Call Center.
- Each component then connects to each other via TCP.

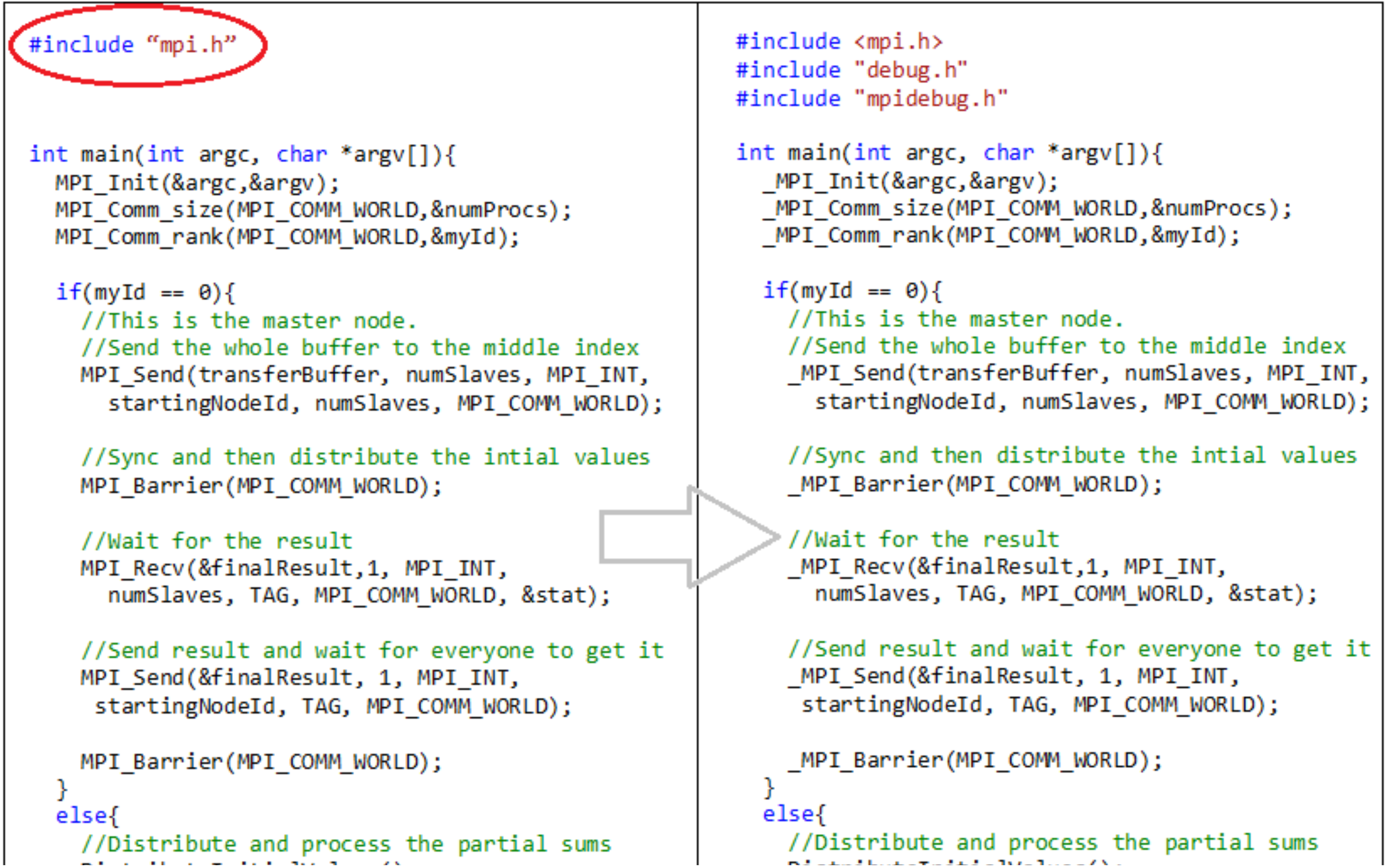
Connection Initiated



MPI Cluster Side

- Include a special mpi.h header file
- MPI calls are caught by wrapper functions
- Upon start up, each node creates a callback connection to The Call Center
- Data passed to MPI functions is sent back.

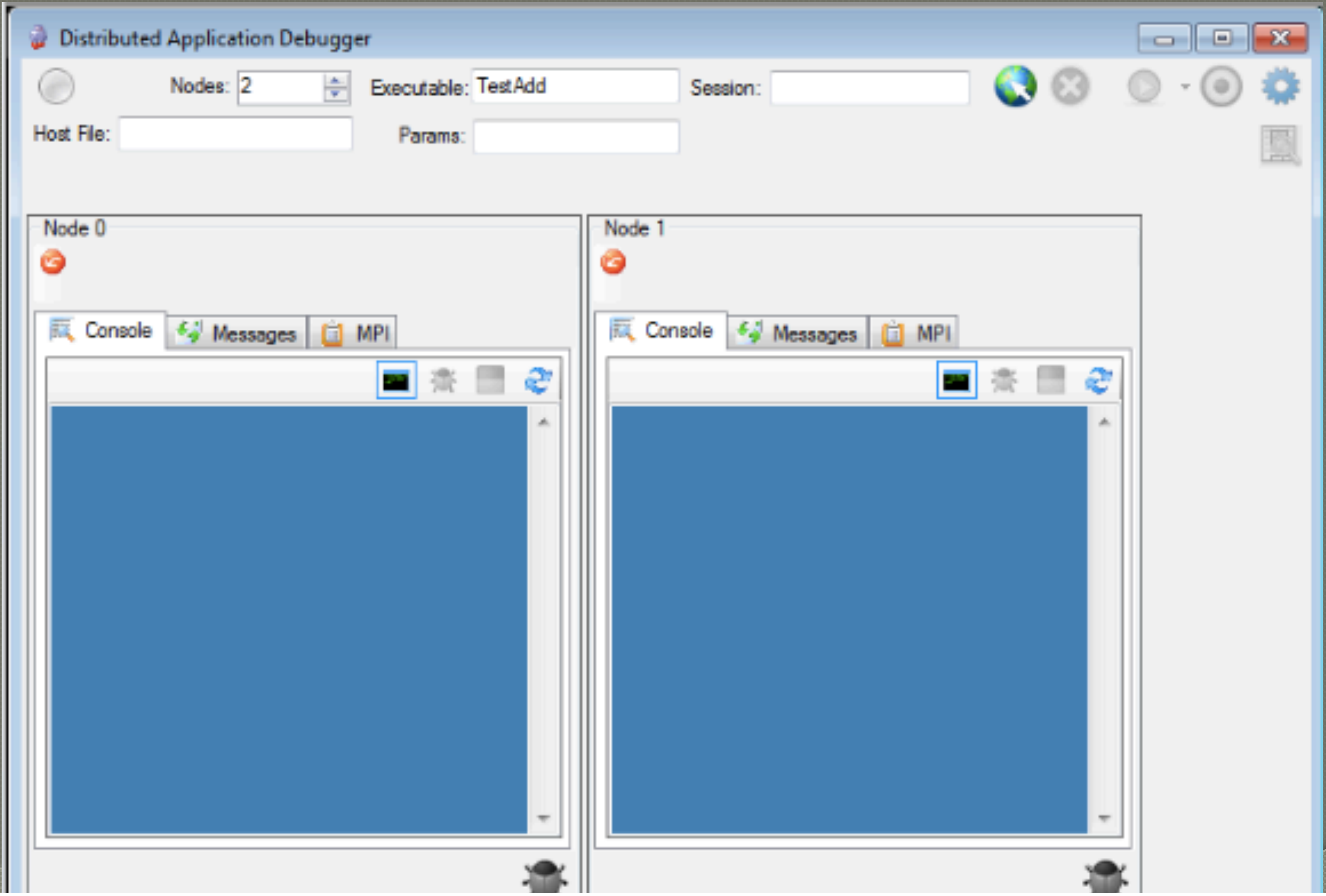
MPI Code Redirected



```
#include “mpi.h”


int main(int argc, char *argv[]){
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numProcs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myId);

  if(myId == 0){
    //This is the master node.
    //Send the whole buffer to the middle index
    MPI_Send(transferBuffer, numSlaves, MPI_INT,
      startingNodeId, numSlaves, MPI_COMM_WORLD);

    //Sync and then distribute the intial values
    MPI_Barrier(MPI_COMM_WORLD);

    //Wait for the result
    MPI_Recv(&finalResult,1, MPI_INT,
      numSlaves, TAG, MPI_COMM_WORLD, &stat);

    //Send result and wait for everyone to get it
    MPI_Send(&finalResult, 1, MPI_INT,
     startingNodeId, TAG, MPI_COMM_WORLD);

    MPI_Barrier(MPI_COMM_WORLD);
  }
  else{
    //Distribute and process the partial sums
    DistributeInitialValues();
```

```
#include <mpi.h>
#include "debug.h"
#include "mpidebug.h"

int main(int argc, char *argv[]){
  _MPI_Init(&argc,&argv);
  _MPI_Comm_size(MPI_COMM_WORLD,&numProcs);
  _MPI_Comm_rank(MPI_COMM_WORLD,&myId);

  if(myId == 0){
    //This is the master node.
    //Send the whole buffer to the middle index
    _MPI_Send(transferBuffer, numSlaves, MPI_INT,
      startingNodeId, numSlaves, MPI_COMM_WORLD);

    //Sync and then distribute the intial values
    _MPI_Barrier(MPI_COMM_WORLD);

    //Wait for the result
    _MPI_Recv(&finalResult,1, MPI_INT,
      numSlaves, TAG, MPI_COMM_WORLD, &stat);

    //Send result and wait for everyone to get it
    _MPI_Send(&finalResult, 1, MPI_INT,
     startingNodeId, TAG, MPI_COMM_WORLD);

    _MPI_Barrier(MPI_COMM_WORLD);
  }
  else{
    //Distribute and process the partial sums
    DistributeInitialValues();
```

The Connected System



Nodes: 2

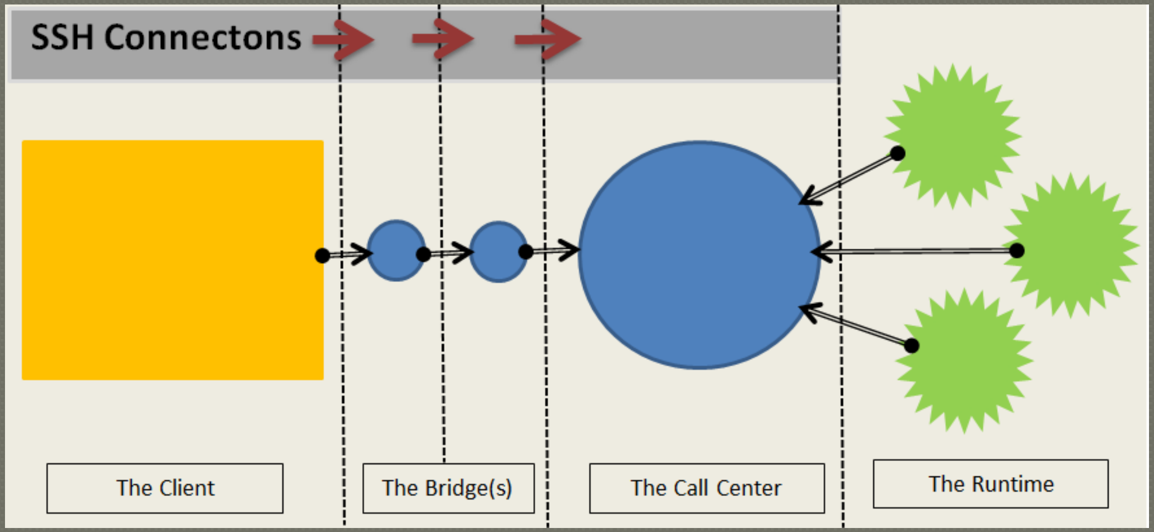
Executable: TestAdd

Session:

Host File:

Params:

The Connected System



Session Modes

An MPI session can be run in 3 modes:

- Play
  - Just run like regular MPI
- Record (Record all messages)
  - Record all messages
- Replay
  - Use recorded messages to play back

Session Modes (Play)

- The Runtime behaves like regular MPI
  - Nothing is saved to disk
  - Nothing is read from disk
  - Messages and parameters ARE sent back to The Client

Session Modes (Record)

- The Runtime
  - Saves messages and parameters to a log file
  - Executes the actual MPI call
  - Saves the result

Session Modes (Replay)

- The Runtime does not execute any real MPI calls.
  - All data is supplied from log files.
  - No actual communication takes place
  - Guarantees the same run as when the log file was recorded

Session Mode (Replay – Mixed)

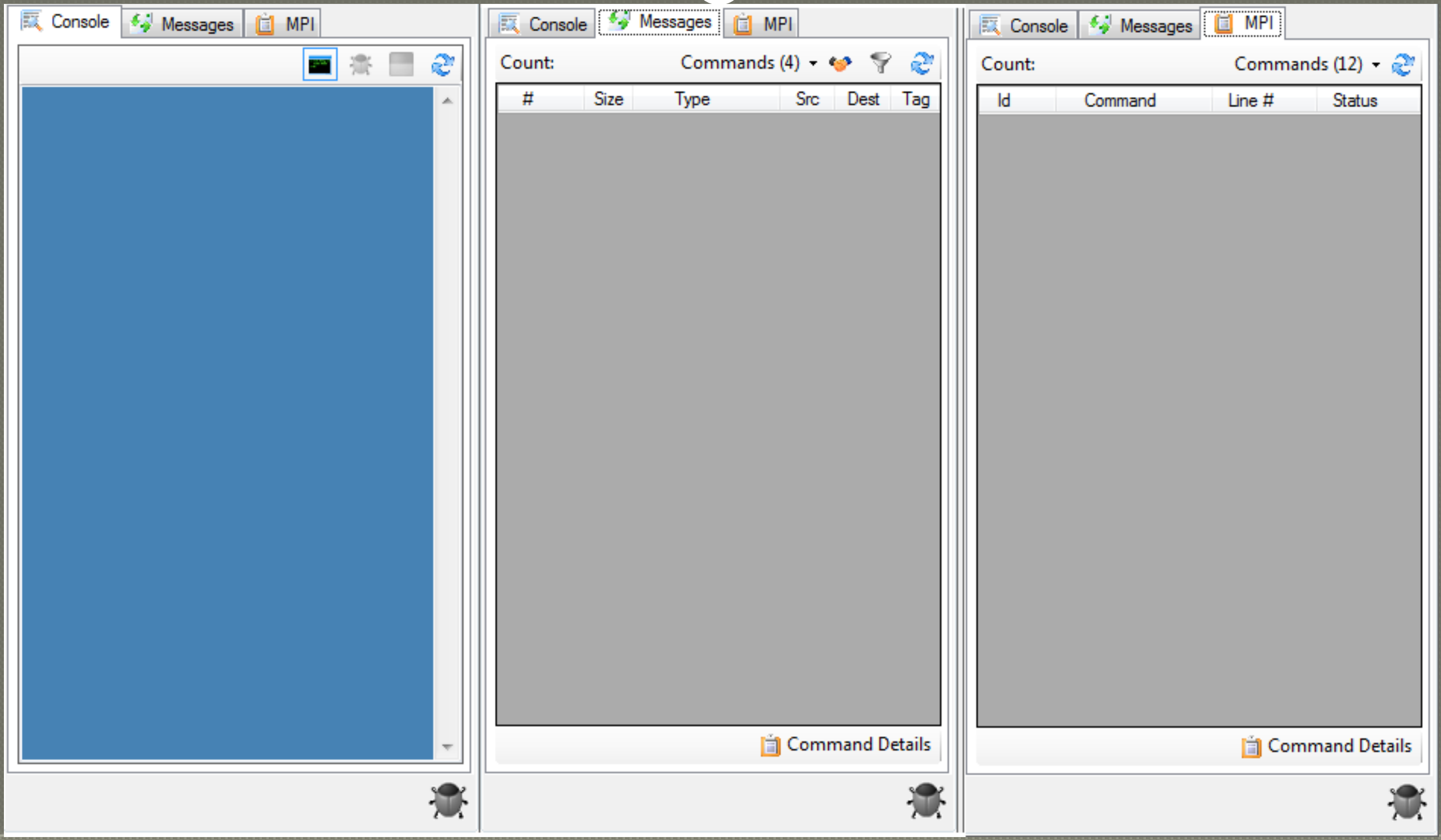
- Mixed mode is special
  - Some processes execute real MPI calls
  - Some replay from log file
    - Sometimes its necessary to execute MPI calls if communicating with someone who is executing real MPI calls; E.g. to avoid buffer overflow
    - Validation is done on real values and log file values

Debugging Data

- The Runtime sends back 2 debugging messages per MPI command
  - A *PRE* message indicating that an MPI command is about to be executed
  - A *POST* message indicating that an MPI command completed
- Console messages are routed per node to the appropriate window.

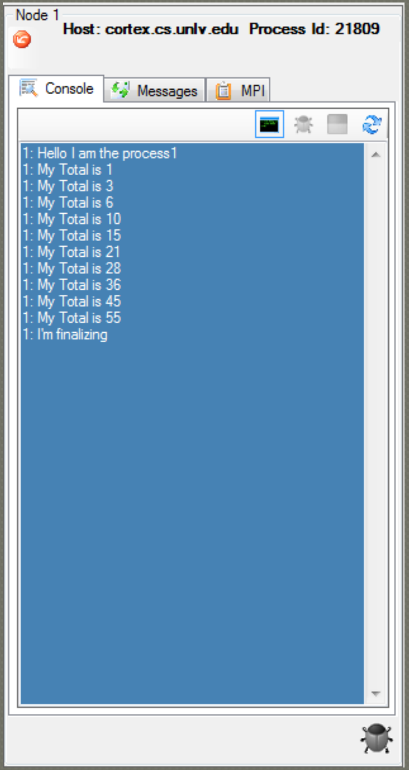
Analyzing Data

- Debugging data gets displayed within the Console, Messages, or MPI tabs



Analyzing Data

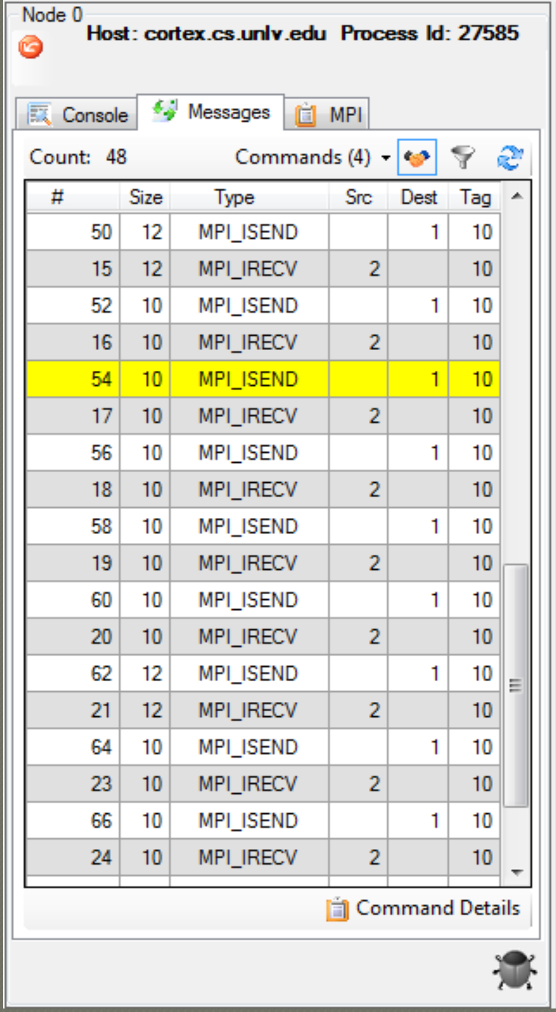
- The Console Tab displays anything that the user’s code wrote to **stdout**.



```
1: Hello I am the process1
1: My Total is 1
1: My Total is 3
1: My Total is 6
1: My Total is 10
1: My Total is 15
1: My Total is 21
1: My Total is 28
1: My Total is 36
1: My Total is 45
1: My Total is 55
1: I'm finalizing
```

Analyzing Data

- The Messages Tab displays messages as they come
- Matches Send/ Receive pairs between nodes.
- Messages without a corresponding Send or Receive message get highlighted in red.

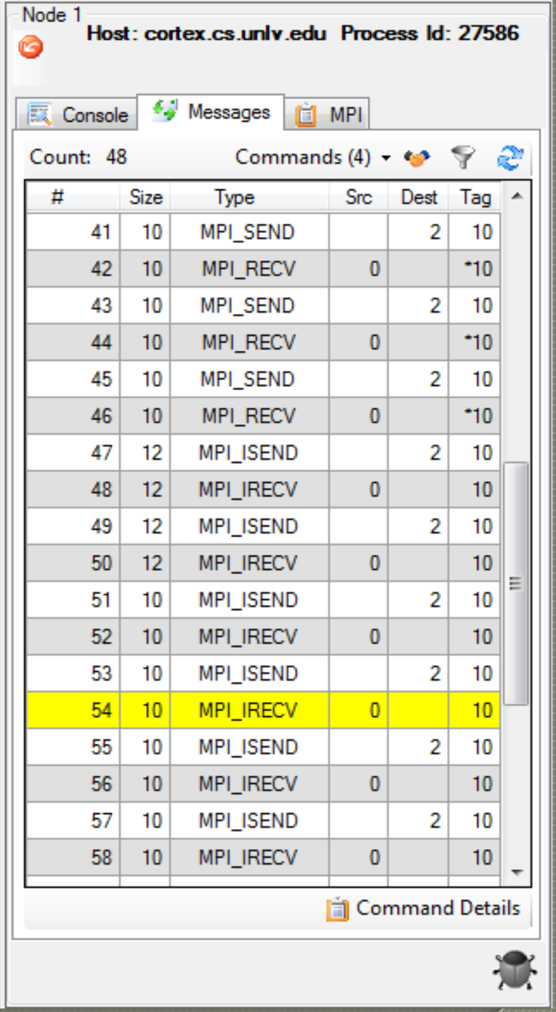


Node 0

Host: cortex.cs.unlv.edu Process Id: 27585

Count: 48 Commands (4)

#	Size	Type	Src	Dest	Tag
50	12	MPI_ISEND		1	10
15	12	MPI_IRECV	2		10
52	10	MPI_ISEND		1	10
16	10	MPI_IRECV	2		10
54	10	MPI_ISEND		1	10
17	10	MPI_IRECV	2		10
56	10	MPI_ISEND		1	10
18	10	MPI_IRECV	2		10
58	10	MPI_ISEND		1	10
19	10	MPI_IRECV	2		10
60	10	MPI_ISEND		1	10
20	10	MPI_IRECV	2		10
62	12	MPI_ISEND		1	10
21	12	MPI_IRECV	2		10
64	10	MPI_ISEND		1	10
23	10	MPI_IRECV	2		10
66	10	MPI_ISEND		1	10
24	10	MPI_IRECV	2		10



Node 1

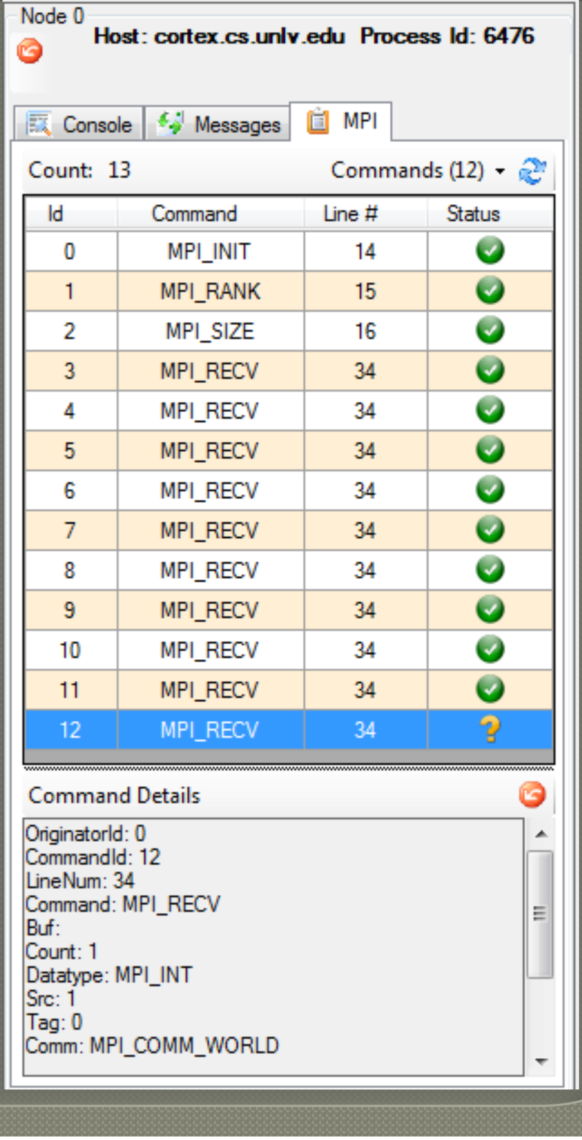
Host: cortex.cs.unlv.edu Process Id: 27586

Count: 48 Commands (4)

#	Size	Type	Src	Dest	Tag
41	10	MPI_SEND		2	10
42	10	MPI_RECV	0		*10
43	10	MPI_SEND		2	10
44	10	MPI_RECV	0		*10
45	10	MPI_SEND		2	10
46	10	MPI_RECV	0		*10
47	12	MPI_ISEND		2	10
48	12	MPI_IRECV	0		10
49	12	MPI_ISEND		2	10
50	12	MPI_IRECV	0		10
51	10	MPI_ISEND		2	10
52	10	MPI_IRECV	0		10
53	10	MPI_ISEND		2	10
54	10	MPI_IRECV	0		10
55	10	MPI_ISEND		2	10
56	10	MPI_IRECV	0		10
57	10	MPI_ISEND		2	10
58	10	MPI_IRECV	0		10

Analyzing Data

- The MPI tab displays all MPI commands
  - in the order they were executed
  - along with their parameters.
- Commands statuses (success, fail, or blocked) are displayed with icons in the Status Column.



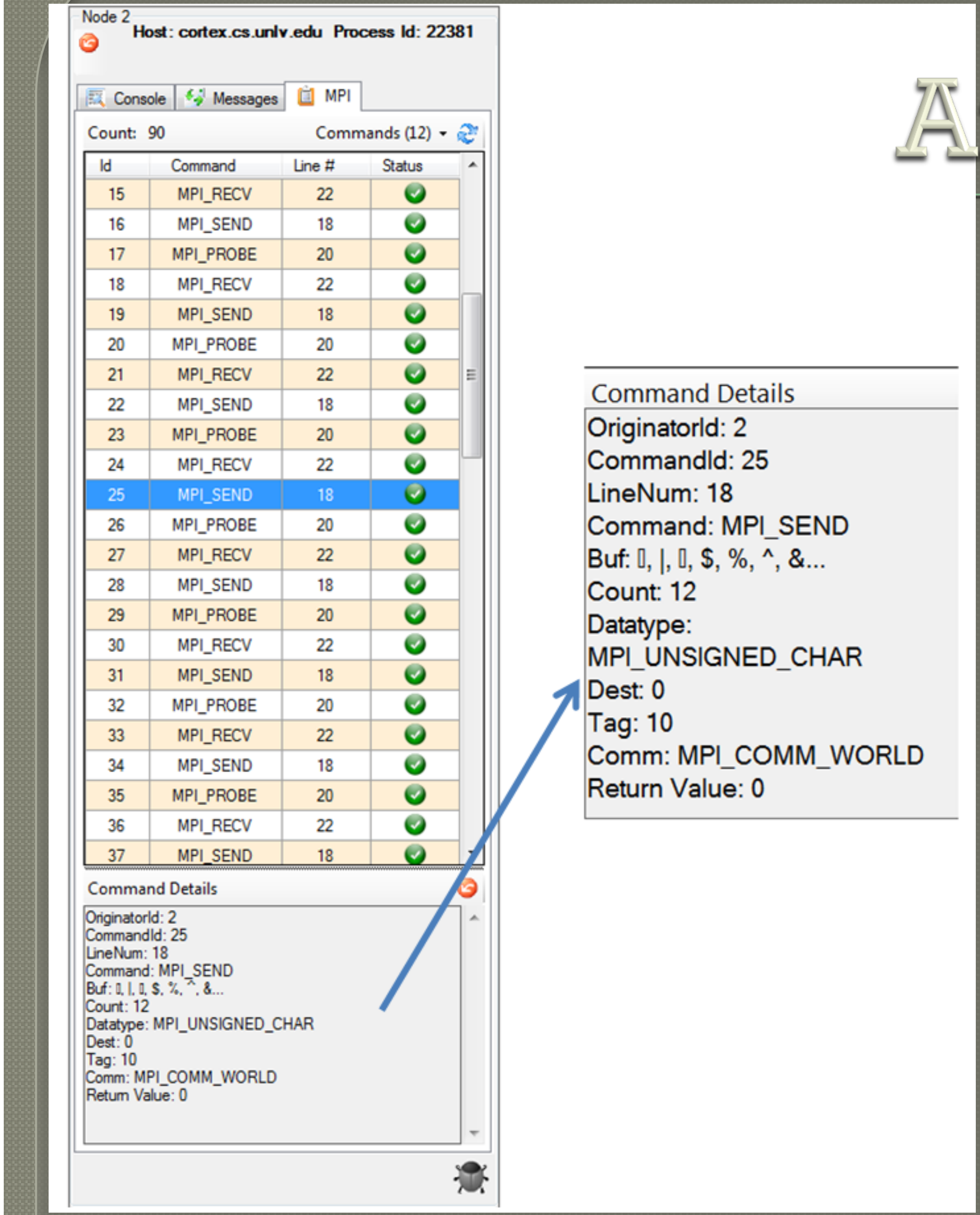
Host: cortex.cs.unlv.edu Process Id: 6476

Count: 13 Commands (12)

Id	Command	Line #	Status
0	MPI_INIT	14	✓
1	MPI_RANK	15	✓
2	MPI_SIZE	16	✓
3	MPI_RECV	34	✓
4	MPI_RECV	34	✓
5	MPI_RECV	34	✓
6	MPI_RECV	34	✓
7	MPI_RECV	34	✓
8	MPI_RECV	34	✓
9	MPI_RECV	34	✓
10	MPI_RECV	34	✓
11	MPI_RECV	34	✓
12	MPI_RECV	34	?

```
OriginatorId: 0
CommandId: 12
LineNum: 34
Command: MPI_RECV
Buf:
Count: 1
Datatype: MPI_INT
Src: 1
Tag: 0
Comm: MPI_COMM_WORLD
```

Analyzing Data



Host: cortex.cs.unlv.edu Process Id: 22381

Count: 90 Commands (12)

Id	Command	Line #	Status
15	MPI_RECV	22	✓
16	MPI_SEND	18	✓
17	MPI_PROBE	20	✓
18	MPI_RECV	22	✓
19	MPI_SEND	18	✓
20	MPI_PROBE	20	✓
21	MPI_RECV	22	✓
22	MPI_SEND	18	✓
23	MPI_PROBE	20	✓
24	MPI_RECV	22	✓
25	MPI_SEND	18	✓
26	MPI_PROBE	20	✓
27	MPI_RECV	22	✓
28	MPI_SEND	18	✓
29	MPI_PROBE	20	✓
30	MPI_RECV	22	✓
31	MPI_SEND	18	✓
32	MPI_PROBE	20	✓
33	MPI_RECV	22	✓
34	MPI_SEND	18	✓
35	MPI_PROBE	20	✓
36	MPI_RECV	22	✓
37	MPI_SEND	18	✓

Command Details

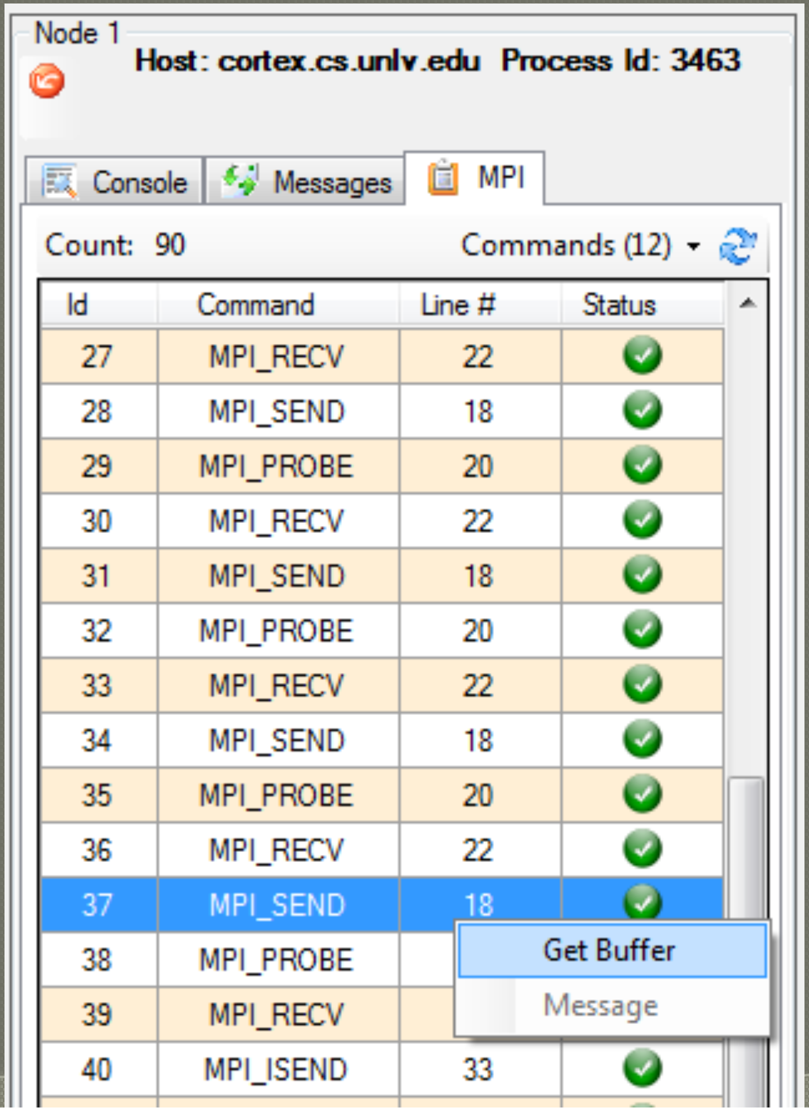
OriginatorId: 2
CommandId: 25
LineNum: 18
Command: MPI_SEND
Buf: ▯, |, ▯, $, %, ^, &...
Count: 12
Datatype: MPI_UNSIGNED_CHAR
Dest: 0
Tag: 10
Comm: MPI_COMM_WORLD
Return Value: 0

Command Details

OriginatorId: 2
CommandId: 25
LineNum: 18
Command: MPI_SEND
Buf: ▯, |, ▯, $, %, ^, &...
Count: 12
Datatype:
MPI_UNSIGNED_CHAR
Dest: 0
Tag: 10
Comm: MPI_COMM_WORLD
Return Value: 0

Analyzing Data

- Buffer values can be requested and inspected.



Node 1

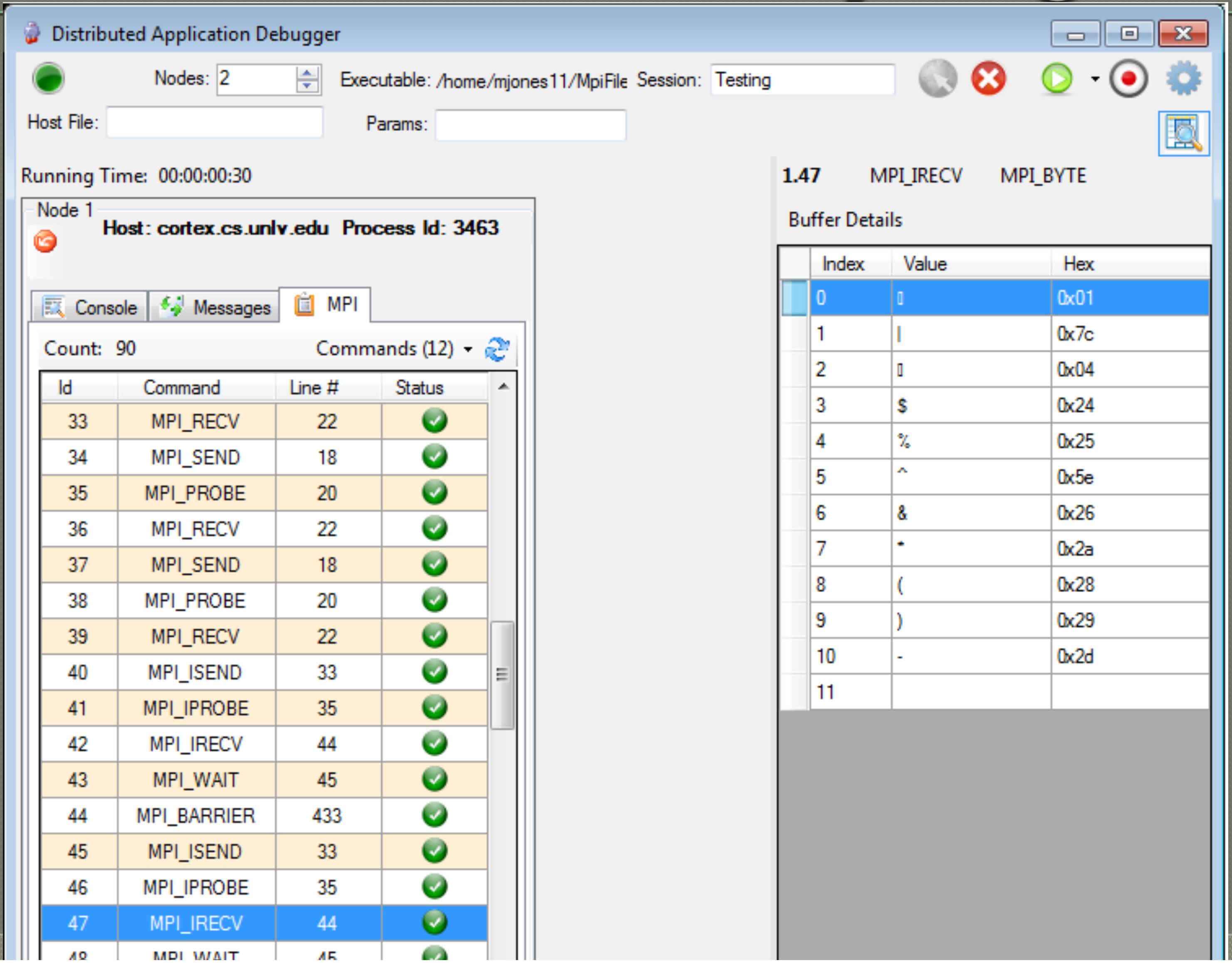
Host: cortex.cs.unlv.edu Process Id: 3463

Count: 90

Commands (12)

Id	Command	Line #	Status
27	MPI_RECV	22	✓
28	MPI_SEND	18	✓
29	MPI_PROBE	20	✓
30	MPI_RECV	22	✓
31	MPI_SEND	18	✓
32	MPI_PROBE	20	✓
33	MPI_RECV	22	✓
34	MPI_SEND	18	✓
35	MPI_PROBE	20	✓
36	MPI_RECV	22	✓
37	MPI_SEND	18	✓
38	MPI_PROBE		
39	MPI_RECV		
40	MPI_ISEND	33	✓

Analyzing Data



Distributed Application Debugger

Nodes: 2 Executable: /home/mjones11/MpiFile Session: Testing

Host File: Params:

Running Time: 00:00:00:30

Node 1

Host: cortex.cs.unlv.edu Process Id: 3463

Count: 90 Commands (12)

Id	Command	Line #	Status
33	MPI_RECV	22	✓
34	MPI_SEND	18	✓
35	MPI_PROBE	20	✓
36	MPI_RECV	22	✓
37	MPI_SEND	18	✓
38	MPI_PROBE	20	✓
39	MPI_RECV	22	✓
40	MPI_ISEND	33	✓
41	MPI_IPROBE	35	✓
42	MPI_IRECV	44	✓
43	MPI_WAIT	45	✓
44	MPI_BARRIER	433	✓
45	MPI_ISEND	33	✓
46	MPI_IPROBE	35	✓
47	MPI_IRECV	44	✓
48	MPI_WAIT	45	✓

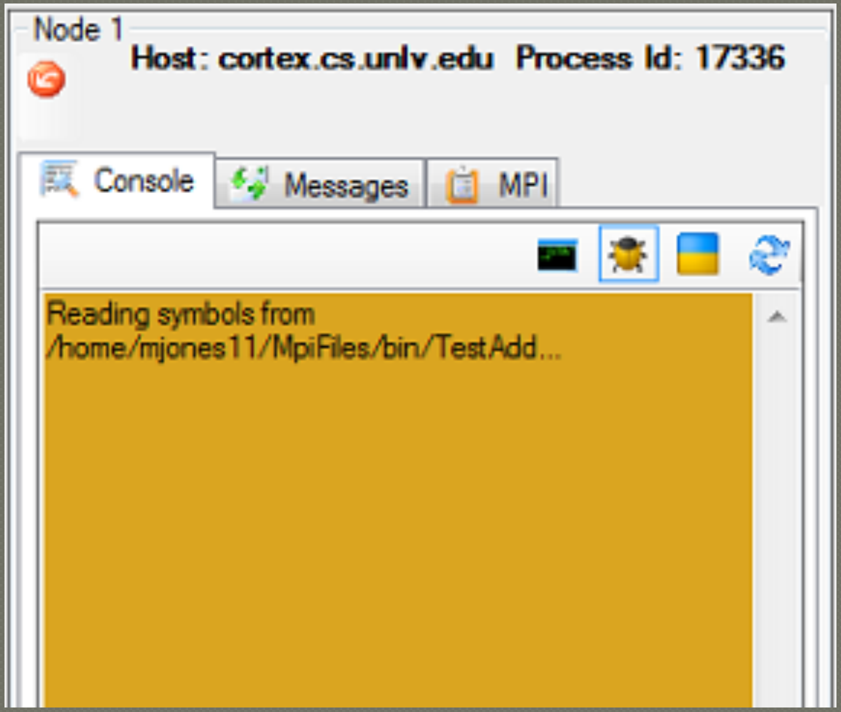
1.47 MPI_IRECV MPI_BYTE

Buffer Details

Index	Value	Hex
0	▯	0x01
1	|	0x7c
2	▯	0x04
3	$	0x24
4	%	0x25
5	^	0x5e
6	&	0x26
7	*	0x2a
8	(	0x28
9	)	0x29
10	-	0x2d
11		

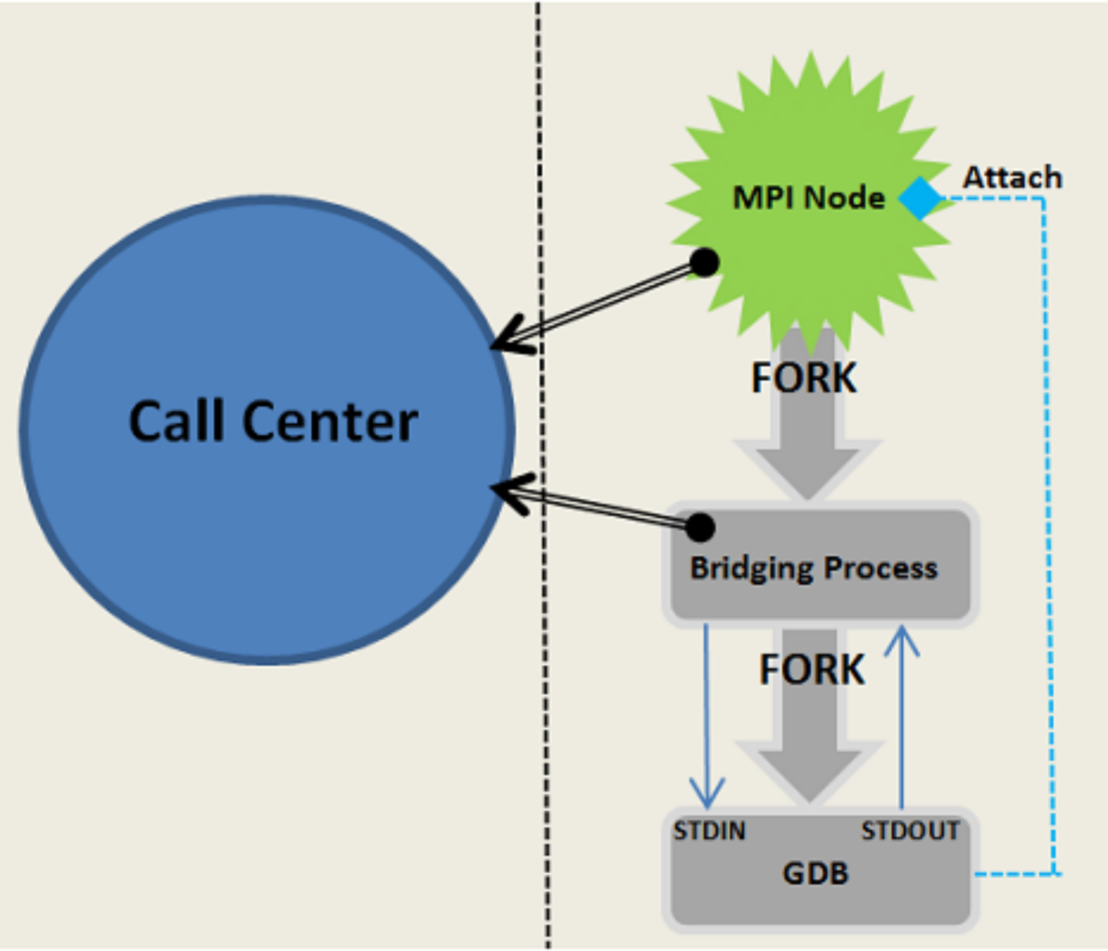
Attaching GDB

- GDB can be attached to any node and controlled with the GDB Control Panel.

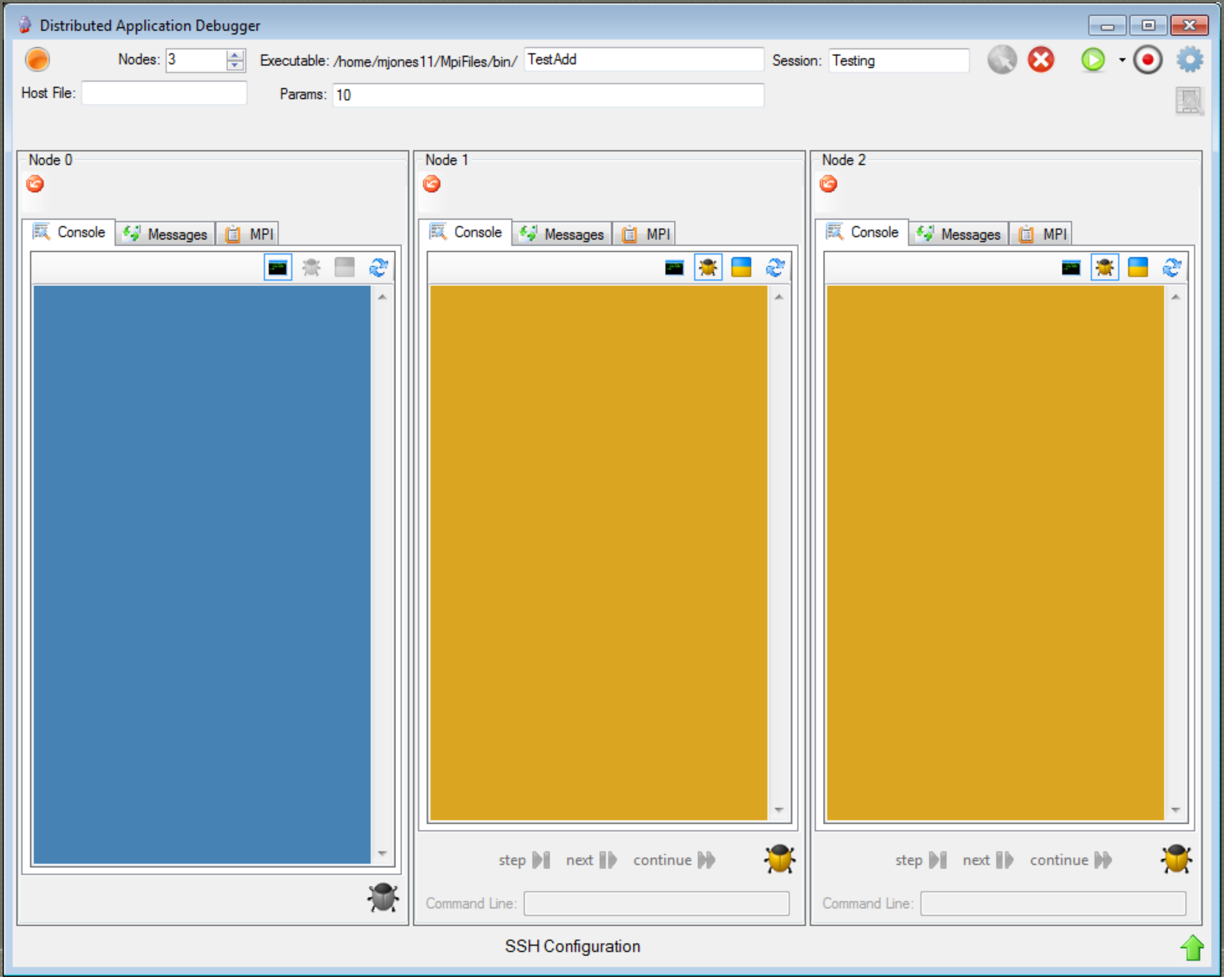


```
Reading symbols from
/home/mjones11/MpiFiles/bin/TestAdd...
```

Attaching GDB



Attaching GDB



Nodes: 3

Executable: /home/mjones11/MpiFiles/bin/ TestAdd

Session: Testing

Host File:

Params: 10

Source Code

- The source code to The Distributed Application Debugger can be found on GitHub at:

- https://github.com/mjones112000/DistributedApplicationDebugger

Questions??