Greg Watson

PTP User/Developer Meeting, Chicago, September 2012

PTP Debugger Troubleshooting

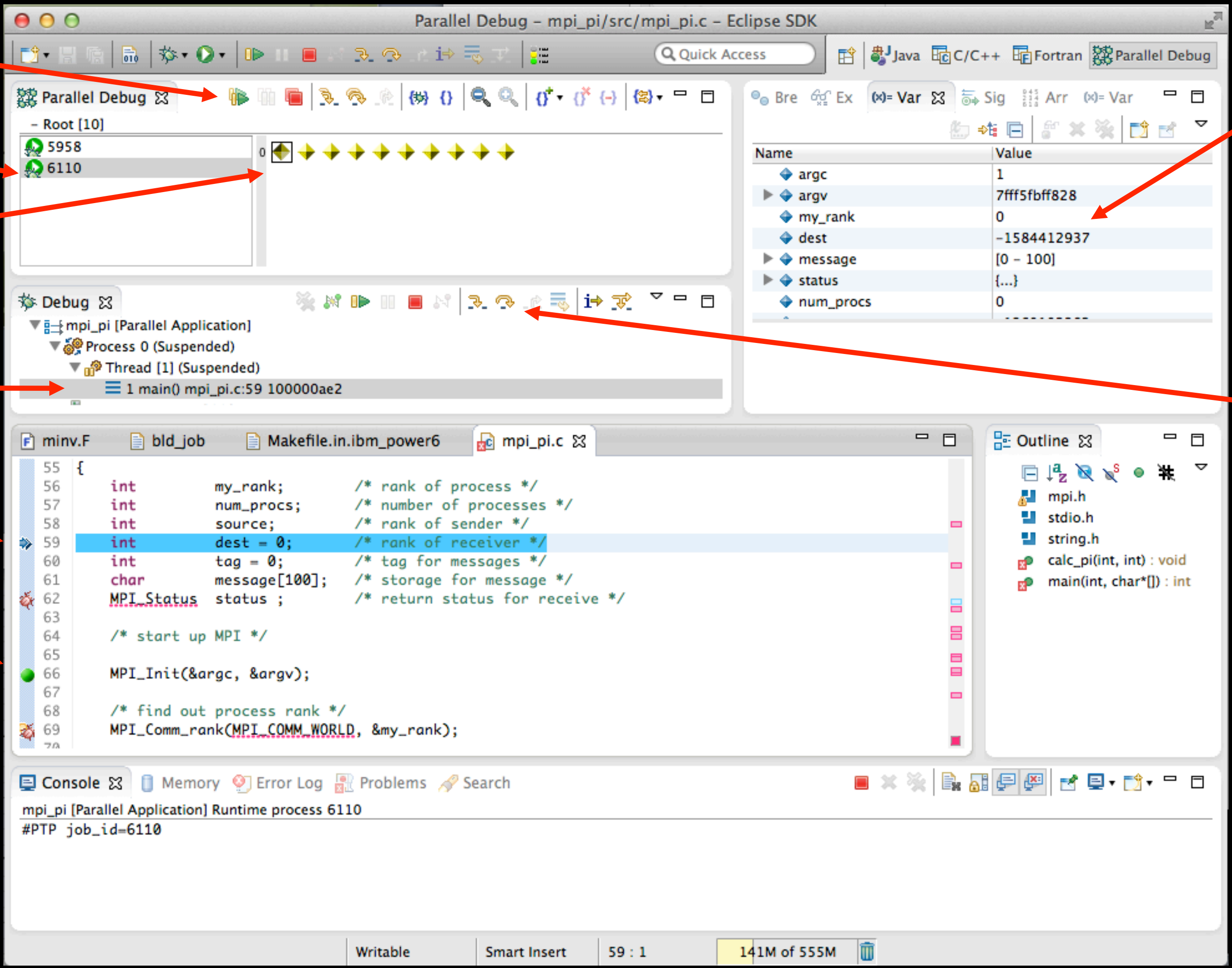
Overview

- Debugger comprises two main components
  - Eclipse client
  - Debug server
- Eclipse client
  - Provides means of interacting with user to perform debug actions
  - Controlling processes
  - Viewing variables
  - Manipulating breakpoints
  - Viewing source code and annotations
- Debug server
  - Scalable Debug Manager (SDM)
  - Multicast/reduction network to control gdb and application instances
  - External program

IBM

Debug Client

- Group commands
- Job ID
- Processes
- Stack frames
- Current line markers
- Breakpoint marker
- Variables
- Process commands



- Root [10]
  - 5958
  - 6110

0

- mpi_pi [Parallel Application]
  - Process 0 (Suspended)
    - Thread [1] (Suspended)
      - 1 main() mpi_pi.c:59 100000ae2

Name	Value
argc	1
argv	7fff5fbff828
my_rank	0
dest	-1584412937
message	[0 - 100]
status	{...}
num_procs	0

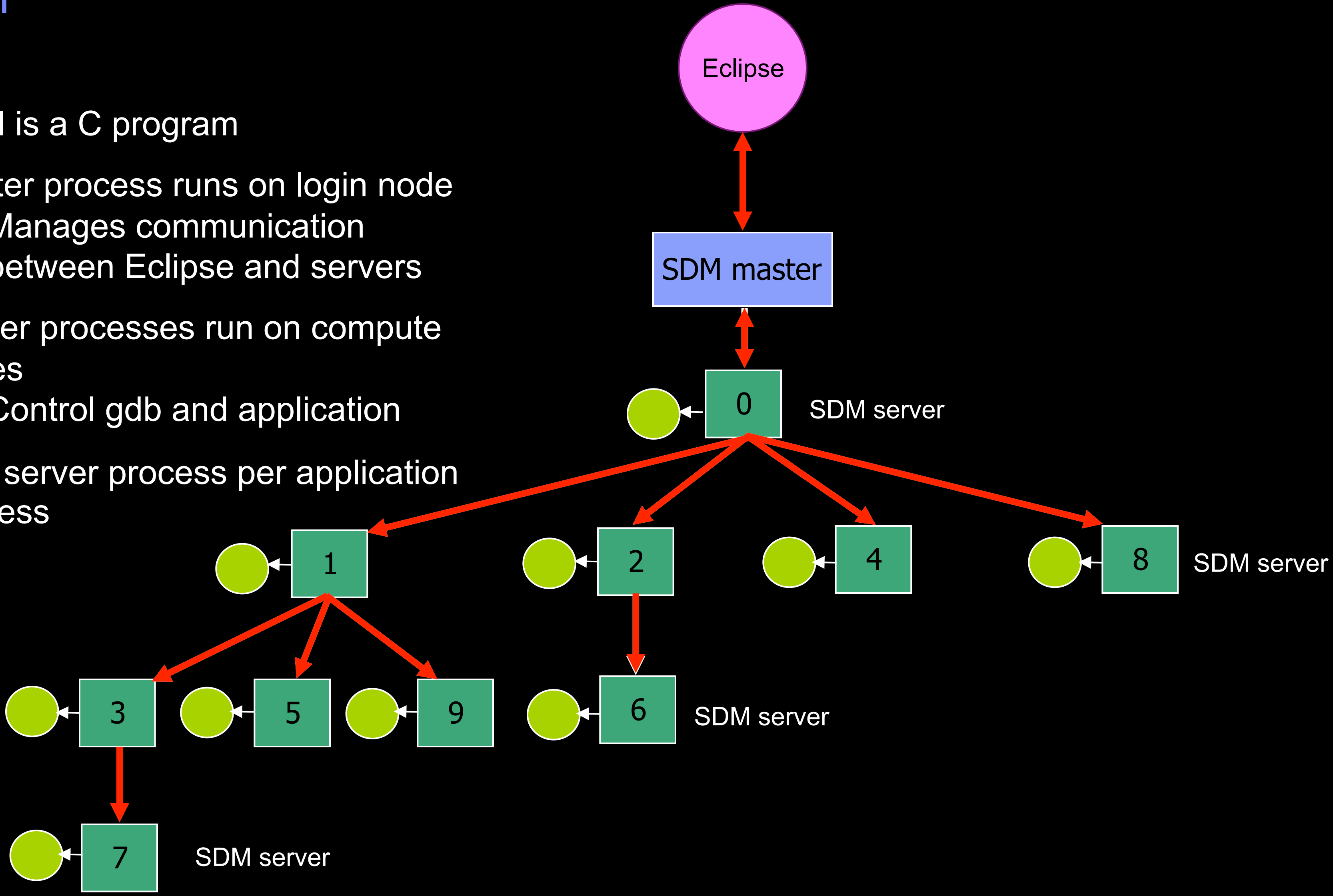
```
55  {
56      int         my_rank;        /* rank of process */
57      int         num_procs;      /* number of processes */
58      int         source;         /* rank of sender */
59      int         dest = 0;       /* rank of receiver */
60      int         tag = 0;        /* tag for messages */
61      char        message[100];   /* storage for message */
62      MPI_Status  status ;        /* return status for receive */
63
64      /* start up MPI */
65
66      MPI_Init(&argc, &argv);
67
68      /* find out process rank */
69      MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

- mpi.h
- stdio.h
- string.h
- calc_pi(int, int) : void
- main(int, char*[]) : int

```
mpi_pi [Parallel Application] Runtime process 6110
#PTP job_id=6110
```

SDM

- SDM is a C program
- Master process runs on login node
  - Manages communication between Eclipse and servers
- Server processes run on compute nodes
  - Control gdb and application
- One server process per application process



Debugger Client Startup

- A debug launch configuration contains
  - Name of executable to debug
  - Path to sdm executable on target system
  - Address to use for sdm connection (session address, normally localhost)
- When the user launches the debugger
  - A socket is created with a random port number bound to it
  - The <submit-interactive-debug> script in the target configuration is invoked
  - The UI waits for an incoming connection on the port by displaying a dialog
- When the connection is established
  - Initialization command is sent to the SDM master
  - If this succeeds, debugger UI is initialized

SDM Startup

- The resource manager is responsible for launching the SDM using the <submit-interactive-debug> script
- Actions required are
  - Launch the SDM server processes onto the compute nodes
  - Generate a routing file containing the rank, hostname, and port number for each server process in the tree
  - Launch the SDM master process passing the session address and port to connect to Eclipse client
- There are currently three target configurations that support debugger launch
  - Open MPI-Generic-Interactive
  - IBM Parallel Environment
  - Torque-Generic-Interactive

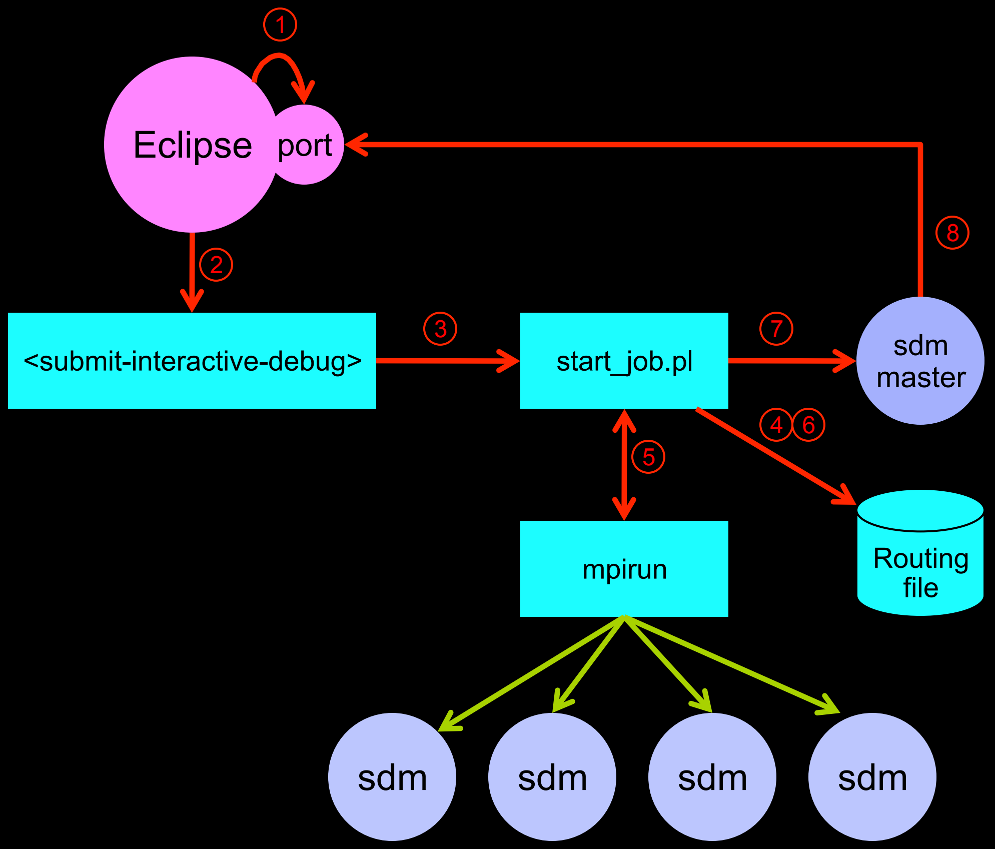
Open MPI-Generic-Interactive

- Uses the script ~/.eclipsesettings/rms/OPENMPI/start_job.pl
- Creates an empty routing file at a known location
- Launches the SDM servers using the command
  - mpirun –mca orte_show_resolved_nodenames 1 -display-map /path/to/sdm
- The output from the mpirun command is parsed, and the rank and hostname information extracted from it
- This is then written to the empty routing file
- The script then starts the SDM master with the appropriate arguments

IBM Parallel Environment

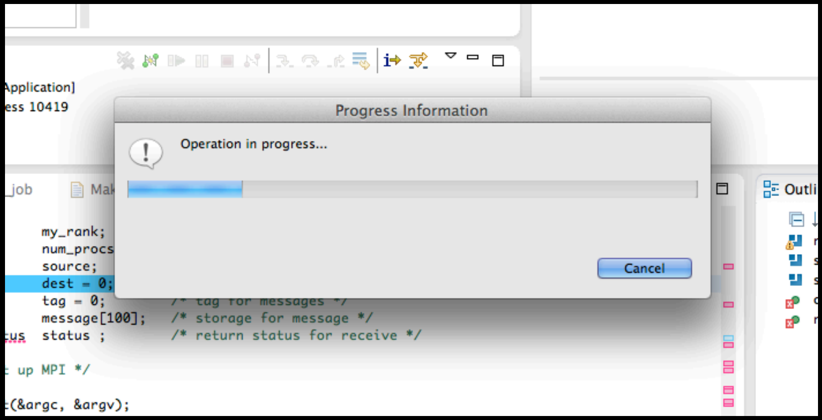
- Uses the script ~/.eclipsesettings/rms/PE/run_pe_app.pl
- Creates an empty routing file at a known location
- Launches the SDM servers using the command
  - poe /path/to/sdm
- Waits for poe to generate an attach.cfg file, then uses this file to generate the routing information
- This is then written to the empty routing file
- The script then starts the SDM master with the appropriate arguments

Debugger Launch



1. Create socket and bind to port
2. Invoke <submit-interactive-debug> script
3. Invoke start_job.pl (or run_pe_app.pl) script
4. Create empty routing file
5. Invoke mpirun (or poe) command to launch SDM servers
6. Generate routing information and write to routing file
7. Invoke SDM master
8. SDM master connects to Eclipse port

Troubleshooting



```
Application]
ess 10419

my_rank;
num_procs
source;
dest = 0;
tag = 0;        /* tag for messages */
message[100];   /* storage for message */
us  status ;    /* return status for receive */

 up MPI */

(&argc, &argv);
```

- Most common symptom is stuck “Operation in progress…” dialog
  - Just indicates that the SDM master has not connected back to Eclipse client
  - One of the other steps must have failed
  - Need to check each in turn
- Launch script fails
  - Make sure that non-debug launch is working first
  - Since they both use the same driver script this should provide some verification that the underlying system is ok
- SDM servers failed to start
  - Use system command to find ‘sdm’ processes
  - Should be the same number of servers as processes being debugged

Troubleshooting (cont…)

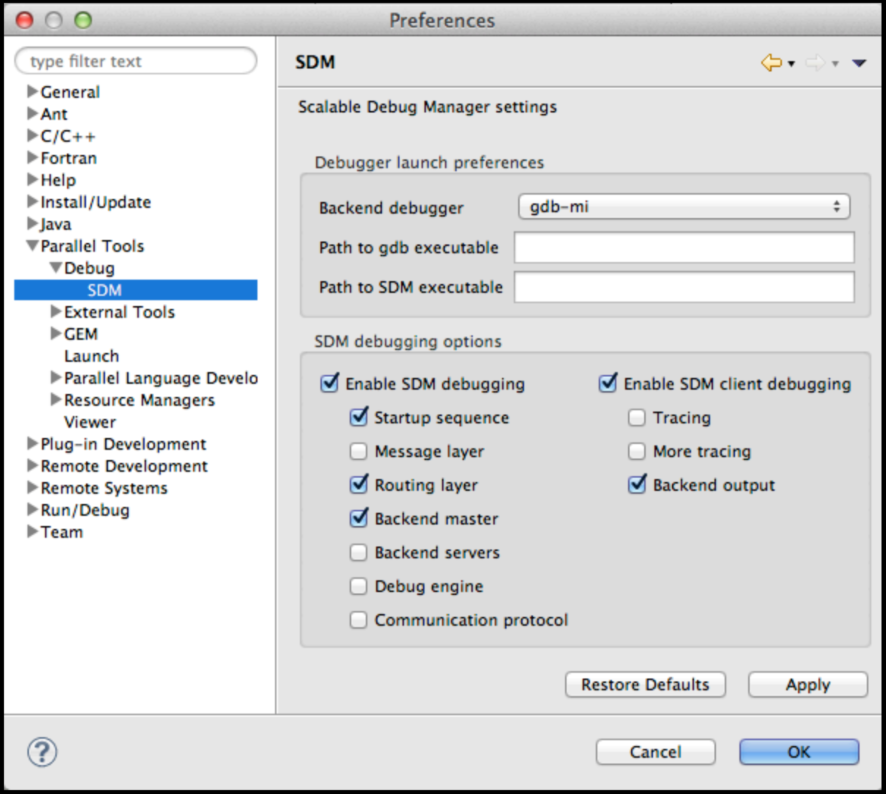
- Could not create routing file
  – Normally located in the same directory as executable being debugged
  – The file is called ‘route_UUID’ where UUID is a long sequence of hex characters and dashes
  – First line should have the number of processes being debugged
  – Each line should contain <rank> <host> <port>
  – Ranks should increase from 0 to N-1
  – Ports should be unique for the same hostname
- SDM master has not started
  – Use ‘ps’ command on login node
  – Should be one ‘sdm’ process with argument ‘—master’ (and others)

Troubleshooting (cont…)

- If all the previous are ok, it may be one of the following
- Routing file must be on a shared filesystem
  - The routing file must be accessible to all sdm processes, including the master
- SDM server processes must be able to communicate with each other
  - Check that it's possible to create sockets between compute nodes
- SDM server processes must be able to communicate with the master
  - Check that it's possible to create sockets between compute nodes and the login node
- SDM master must be able to connect to Eclipse client
  - An ssh tunnel is used to communicate between SDM master and Eclipse client
  - Make sure ssh tunnels are allowed
  - Make sure session address on debugger tab is "localhost"

SDM Debug Preferences

- Enable SDM debugging if you are still having problems
- Startup sequence
  - Initial sdm startup
- Message layer
  - Messages sent between sdm processes
- Routing layer
  - Reading routing table
- Backend master
  - Commands executed by master
- Backend servers
  - Server specific messages
- Debug engine
  - Commands sent to gdb
- Communication protocol
  - Low level protocol



SDM

Scalable Debug Manager settings

Debugger launch preferences

Backend debugger: gdb-mi

Path to gdb executable:

Path to SDM executable:

SDM debugging options

☑ Enable SDM debugging	☑ Enable SDM client debugging
☑ Startup sequence	☐ Tracing
☐ Message layer	☐ More tracing
☑ Routing layer	☑ Backend output
☑ Backend master	
☐ Backend servers	
☐ Debug engine	
☐ Communication protocol	