



Scalable Tools Communication Infrastructure

Scalable Debugging Infrastructure: A Case Study

Greg Watson
IBM Research

HPCSW 2008

© 2008 IBM Corporation

Overview

- Parallel Tools Platform Debugger
- Current Design
- Design Issues and Limitations
- A New Approach
- Conclusion

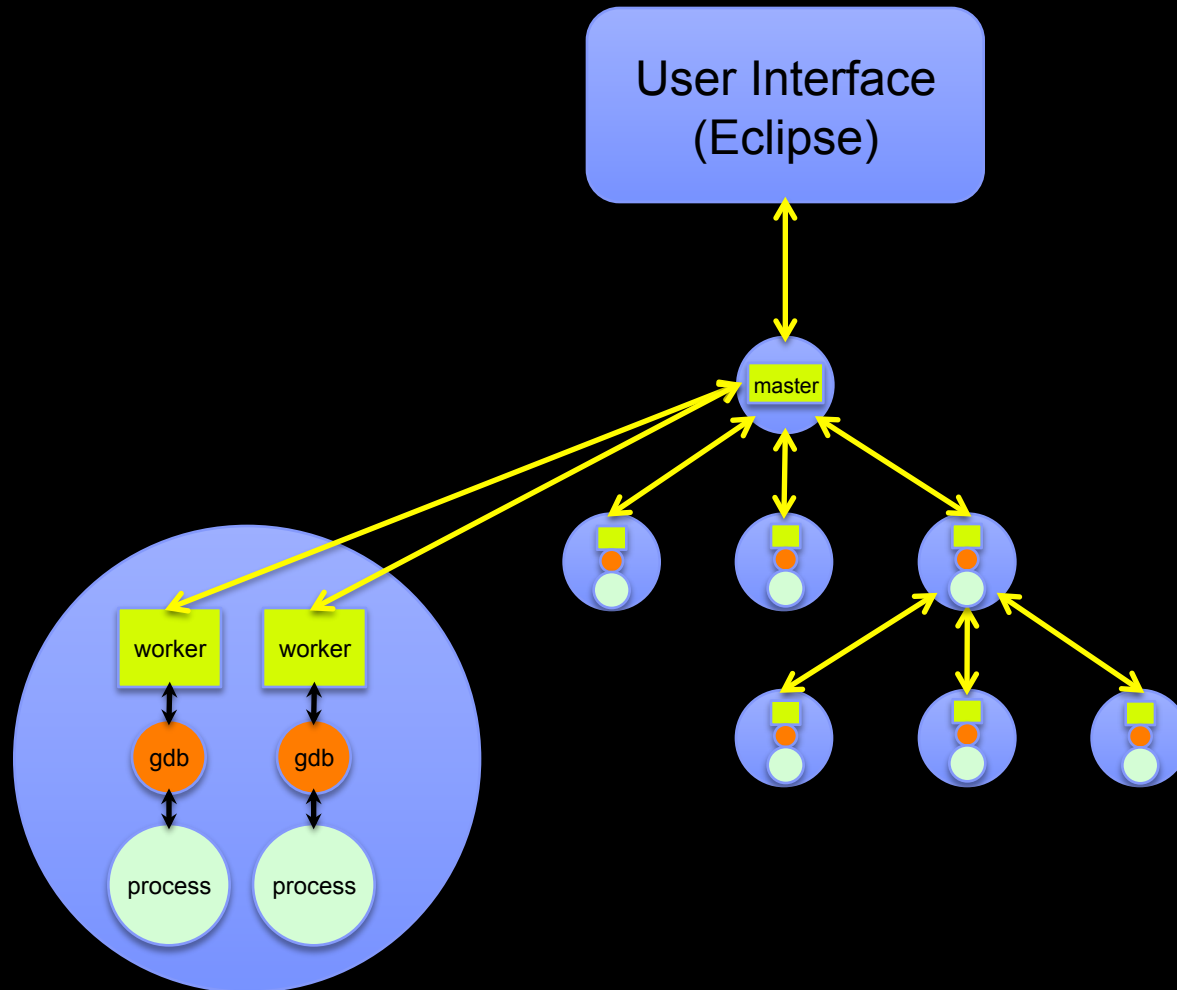
Parallel Tools Platform Debugger

- Ideal for cluster type architectures
- Common debug operations
 - Suspend, resume, step
 - Set breakpoints
 - View variables
 - Etc.
- Operations can be applied to single processes or groups of processes; results are automatically aggregated
- Eclipse is used as user interface; full integration
- Debugger can be used as platform for research into scalability issues

Current Design

- Client/server architecture
- Client is Eclipse
- Server is an MPI program
 - Uses MPI program to debug MPI program
- Server comprises a master process and many worker processes
 - Master communicates with Eclipse client
 - Workers control application processes and also act as routing nodes for communication
 - Worker topology computed at runtime
- GDB is used as backend debug engine (pluggable)
- Launching under control of debugger is the only startup mode supported currently (could add attach if required)

Current Design (cont...)



Design Issues and Limitations

- No common mechanism for resource management and launching debuggers
 - Each runtime system requires specific interface
 - Debugger must have knowledge of runtime system
 - Starting application under debugger control is very architecture specific
 - Hard to map application to debug processes when attaching
- Debugger startup may not scale well
- Debugger must compute topology at runtime
- Debugger must implement broadcast/reduction (no asynchronous collectives)

Design Issues and Limitations (cont...)

- Debugger responsible for managing communication for aggregation
- Hard to deal with debugger failure
- No way to detect other failure events (e.g. node failure)
- What if it takes days to reach point of error in application?
- What if application size changes (e.g MPI_Spawn)?

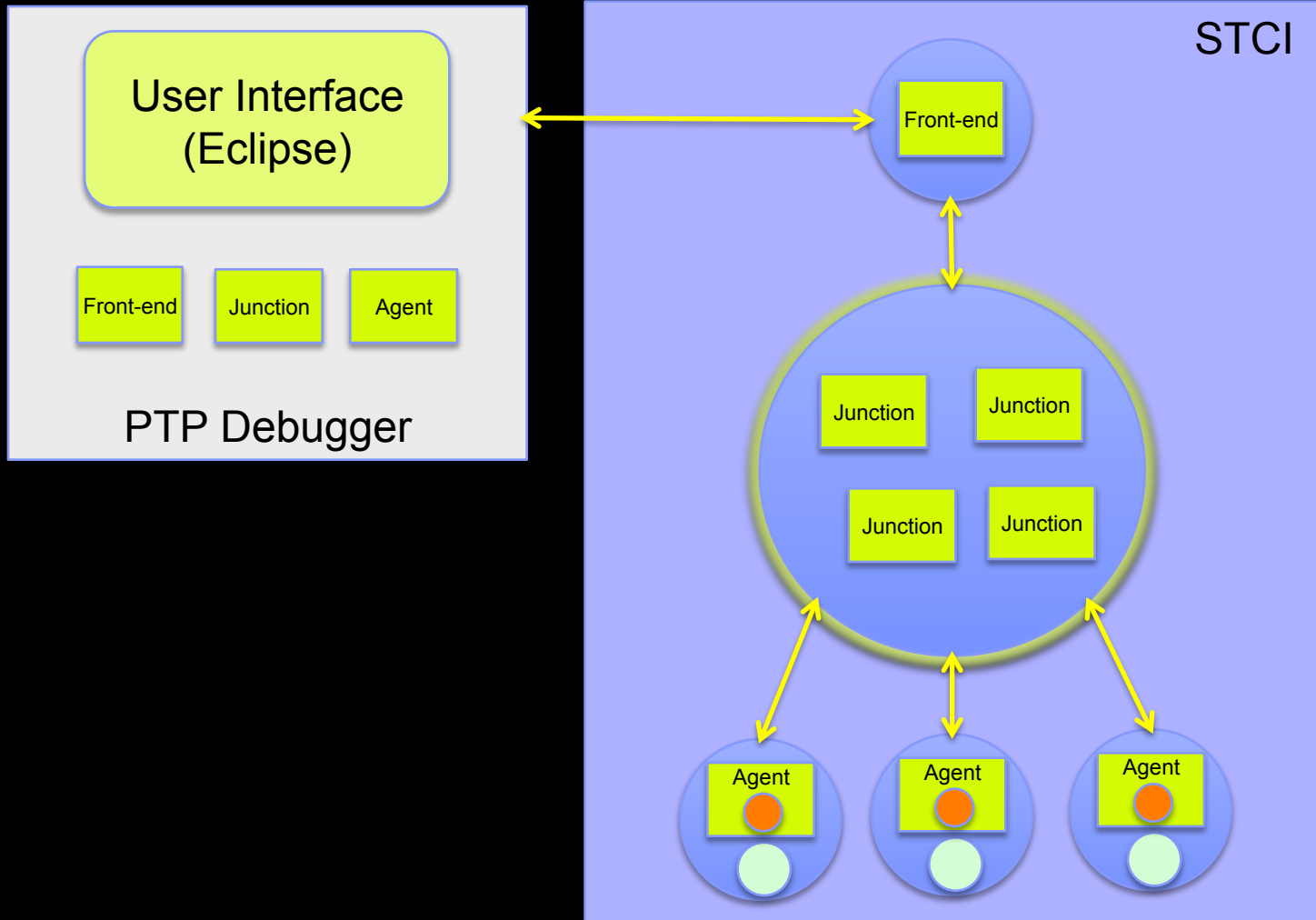
A New Approach

- Debugger comprises four parts
 - User Interface
 - Front-end
 - Junctions
 - Agents

A New Approach (cont...)

- User Interface
 - Unchanged
- Front-end
 - Manages communication between UI and rest of debugger
 - Supports disconnect/reconnect
 - Is notified when failures occur
- Junctions
 - Implement aggregation technique
 - Do not need to know anything about communication
- Agents
 - Know how to debug single application process
 - No other architecture-specific activities required

New Design (cont...)



Advantages

- All system specific activities handled automatically
 - Resource allocation
 - Debugger process startup/shutdown
 - Application process startup/shutdown
- Ability to detach/reattach for long running jobs
- Receives notification of important events (e.g. failure)
- Junctions (and hence debugger behavior) can be easily modified e.g. for different problem sizes
- Topology is managed automatically
- Focus is on core functionality rather than implementation details

Conclusion

- Developing parallel debuggers that scale is complicated
- It is made even more difficult by having to deal with system specific details
- Porting from one system to another becomes a major undertaking
- As a result, there are few parallel debuggers
- STCI will significantly simplify the design of scalable debuggers and reduce the effort required to support new architectures