On-Line Monitoring Support in PVM and MPI

Roland Wismüller

LRR-TUM
Institut für Informatik
Technische Universtität München
Germany

e-mail: wismuell@in.tum.de

Introduction

- Topic of this talk:
  - comparison between PVM and MPI from a tool developer's point of view
  - focus on features supporting on-line monitoring and tools
- Background:
  - OMIS: On-line Monitoring Interface Specification
    - allows on-line tools to observe and manipulate the execution of parallel and distributed applications
  - OCM: implementation of OMIS for PVM on NOWs
- Occasion for this work:
  - porting the OCM from PVM to MPI (i.e. `mpich`)

Basic Concepts

- Object based approach
  - observed system is a set of objects
  - OMIS offers services to observe and manipulate objects
- Tools specify event / action relations
  - event specifies when something has to be done (may be empty)
  - actions specify what has to be done
  - actions are autonomously executed by the monitoring system
- Location transparency
- Scalability
- Extensibility

OMIS Request Examples

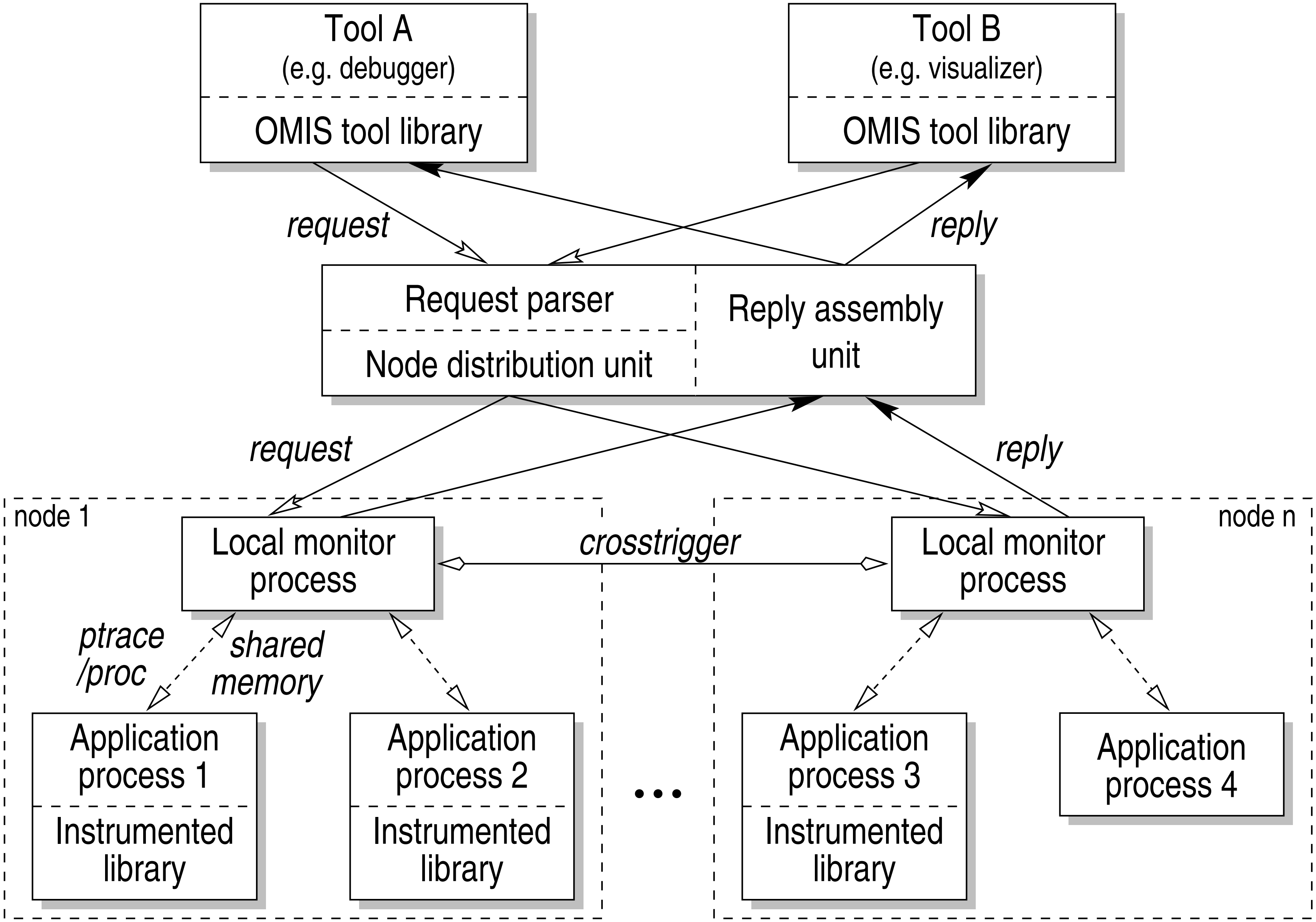
```
thread_reached_addr([p_14,p_52], 0x1234):
       thread_stop([]) thread_get_backtrace($thread, 0)
```

- When (a thread in) one of the specified processes reaches the breakpoint, stop the whole application and return the stack backtrace

```
thread_has_started_lib_call([p_21], "pvm_send"):
       counter_add(counter_1, $par5)
```

- When (a thread in) process p_21 starts a send call, add the message size (event context paramerter par5) to a counter

The OMIS Compliant Monitoring System OCM



Interactions Between OCM and Target System

Interactions with operating system

- ➥ Information on specific nodes
- ➥ Information on (and modification of) specific processes
- ☞ No notion of an application
- ☞ Relatively inefficient

Interactions with programming library

- ➥ Start-up and initialization
- ➥ Access (global) state of target application and programming library
- ➥ Efficient detection of events related to the programming library

Startup and Initialization

Two possible scenarios

- Tool starts the target application (via the OCM)
- Tool (and OCM) connects to an already running application

Starting an application

- We could use `pvm_spawn` or `MPI_Comm_spawn_multiple`
- But:
  - processes now have a PVM/MPI parent $\Rightarrow$ different behavior
  - not applicable to MPI-1
- For PVM, we use `fork` / `exec`
- For MPI, we use a startup-option already available with `mpirun`

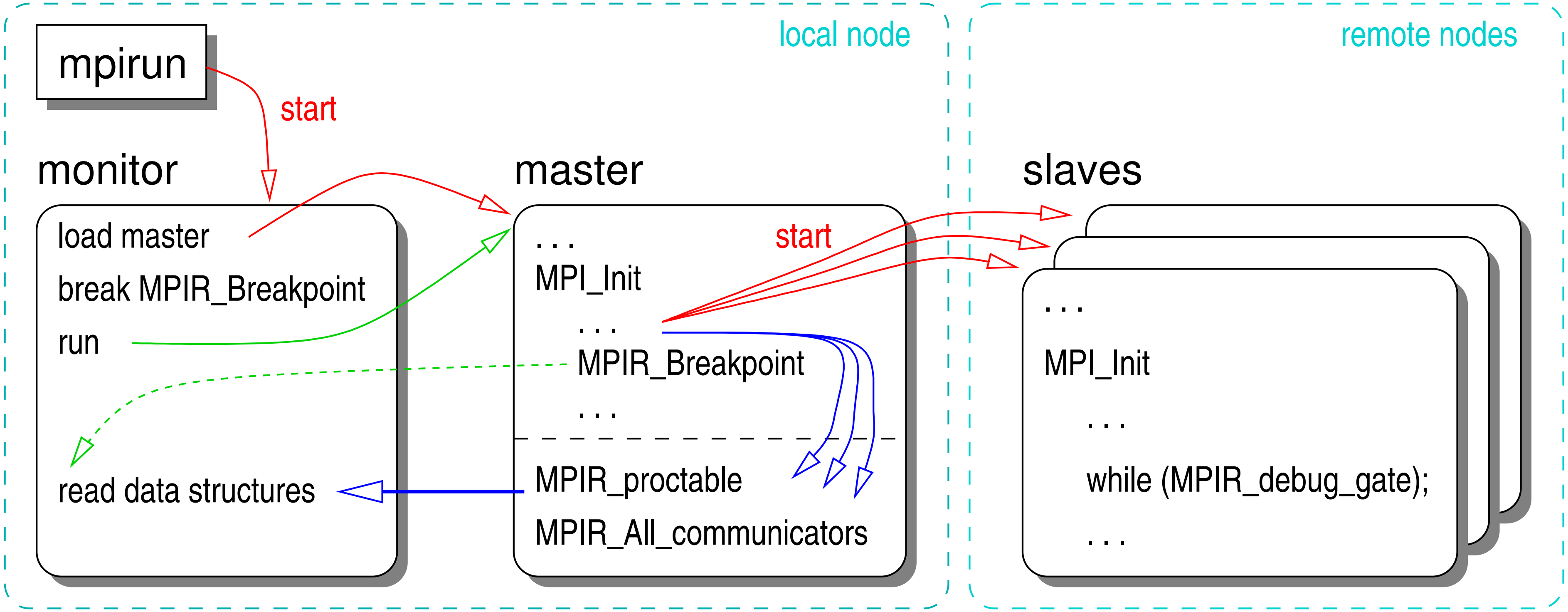
Startup and Initialization

Connecting to an application

- The OCM needs at least the following information on the application:
  - list of nodes (hostname)
  - list of processes (node, Unix PID, path to executable, global ID)
- All other information can be obtained via the OS
- In PVM: `pvm_config`, `pvm_tasks`
  - only possible if at least one OCM process is PVM task
  - application behavior may change due to different results of `pvm_tasks`
  - TIDs change w.r.t. unmonitored execution
- With MPI: use built-in mechanism of `mpirun`

Startup and Initialization

Startup in `mpirun`



- Monitor immediately controls master process
- Monitor can read information on slaves from master's memory
- No easy way to get global IDs

Access to State Information

Relevant information

- (Configuration of target application, i.e. nodes and processes)
- State of communication subsystem, e.g. message queues, waiting processes
- State of other objects offered by programming model, e.g. groups

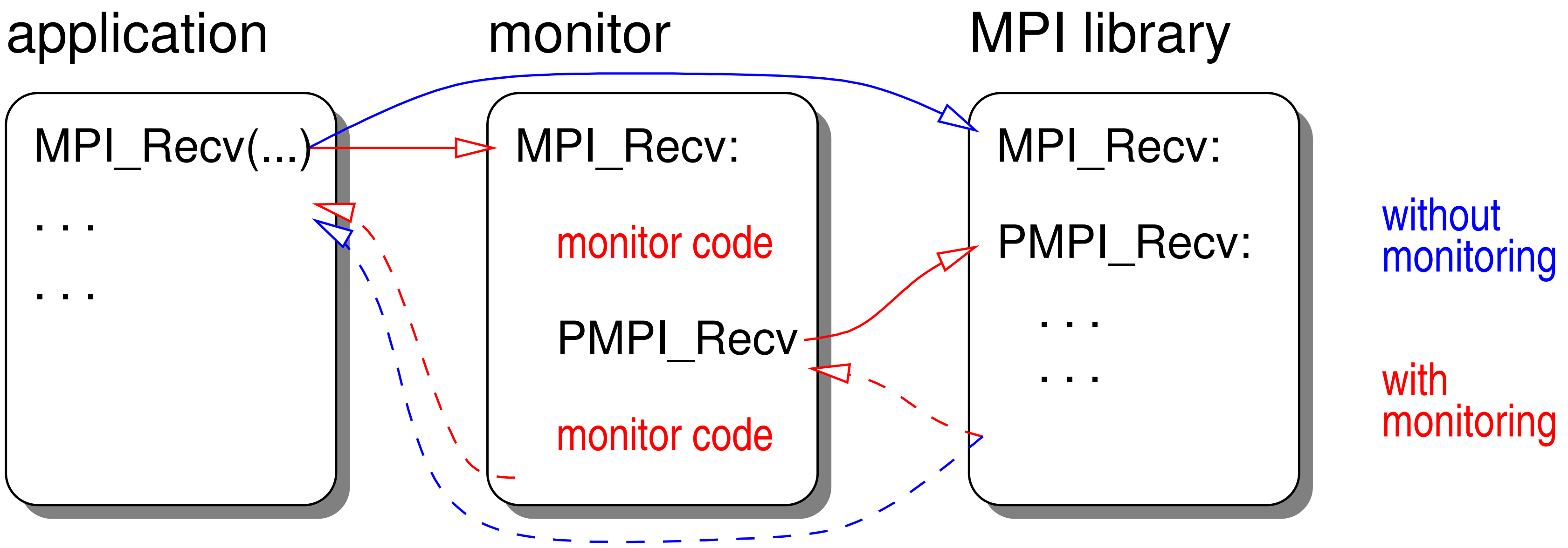
Support for monioring

- No sufficient support in PVM / MPI
  - Have to use undocumented interfaces / data structures within the library
  - Monitoring system depends on specific library implementation

Event Detection

Library calls

- Profiling interface in MPI



- With PVM: use binary wrapping, i.e. rename routines in binary library

Event Detection

Process creation

- ➥ OCM must be able to ‘catch’ the new process before it starts executing
- ☞ Not sufficient to monitor the library call for process creation
- ⊳ In PVM: tasker interface
  - ➥ tasker is a PVM task that is notified by the PVM daemon and then has to create a (local) task
  - ➥ can use OS mechanisms to control the new process
  - ☞ all OCM processes must be PVM tasks
  - ☞ tasker must not terminate before its children
  - ☞ new tasks do not yet have a PVM TID
- ⊳ In MPI: slaves are blocked in an endless loop inside `MPI_Init`
  - ☞ code executed prior to `MPI_Init` cannot be monitored

Using PVM/MPI Inside the OCM

Requirements of OCM

- Handler-driven receive operations
  - OCM must not block; polling is too expensive
  - neither PVM nor MPI offers this feature
  - ☞ need to manipulate IP socket to get a signal
  - ☞ slightly more easy in PVM (`pvm_getfds`)
- Independently started processes (tools) must be able to connect to a running OCM
  - possible in PVM, but not in MPI (even MPI-2)
  - MPI is not intended for distributed applications

Conclusions

Consequences for OCM

- PVM version must be a PVM application and can use PVM internally
- MPI version need not be an MPI application and cannot use MPI
- Communication layer with two implementations: PVM, TCP/IP sockets

Support needed for on-line monitoring

- List of nodes (hostname)
- List of processes (node, Unix PID, path to executable, global ID)
- Means to intercept process creation
- Second entry point for each library function
- Routines to read (modify) the state of objects of the programming model
- Interfaces should be usable for processes not being part of the application