Profiling Programs under Condor: Prototyping the Tool Daemon Protocol
Ana Cortés & Miquel A. Senar
Paradyn/Condor week 2002
Contributors

• Barton Miller
• Vic Zandy
• Philip Roth
• Brandon Schendel
• Andrew Bernat

• Miron Livny
• Derek Wright
• Todd Tannenbaum
• Doug Thain
Abstract

• We have made changes in Condor and Paradyn to make them inter-operable.
• Therefore, a job submitted to Condor can be monitored by Paradyn.
• The modifications included in both counterparts will be sketched in this presentation.
• This work is the first step in prototyping the Tool Daemon Protocol (TDP).
Outline

• TDP + Paradyn + Condor background
• Combining Paradyn and Condor: the TDP prototype
• Submitting and starting a monitored job
• Problems & Side effects
• Current status
• Work in progress
TDP (Tool Daemon Protocol)

TDP issues:
- Application creation & starting
- Application control & monitoring
- Exchange of information between remote process manager and tool daemon about application state
- Control of standard I/O
Condor mode of operation

job submit

Match Maker

schedd

startd
Condor mode of operation

Job submit

Job

Match Maker

Claiming & Activation

Execution protocol

Home File System

Startd

Schedd

Shadow

Starter

Creates the execution environment.

fork

fork

fork

fork
Paradyn: “create process” mode

- the front-end creates the paradynd using either fork (local machine) or rsh (remote machine)
Paradyn: “create process” mode

- the front-end creates the paradynd using either fork (local machine) or rsh (remote machine)
- the paradynd connects to the front-end
- creates the application process
Paradyn: “create process” mode

- The front-end creates the paradynd using either fork (local machine) or rsh (remote machine).
- The paradynd connects to the front-end.
- Creates the application process.
- The paradyn run-time library is loaded into the application process.
- Paradynd parses the executable to discover symbols and find potential instrumentation points.
- A connection is established between the application and the paradynd.
Paradyn: “attach” mode

there is an already running application that wants to be monitored
there is an already running application that wants to be monitored

• the front-end creates the paradynd
• the paradynd connects to the front-end.
Paradyn: “attach” mode

- **Paradyn Front-End**
  - fork/rsh

- **paradynd**
  - attach

- **Appl.**

**there is an already running application that wants to be monitored**

- the front-end creates the paradynd
- the paradynd connects to the front-end.

- the paradynd attaches to the application process,
- the **paradynd run-time library** is loaded into the application process
- **paradynd parses** the executable to discover symbols and find potential instrumentation points
- a connection is established between the application and the paradynd
Combining Paradyn and Condor: the TDP prototype

Paradyn Front-End

job submit

Match Maker

schedd

startd
Combining Paradyn and Condor: the TDP prototype

Paradyn Front-End

job submit

Match Maker

schedd

fork

shadow

fork

starter

new job

Home File System

paradynd

Appl.
Combining Paradyn and Condor: the TDP prototype

- Condor is responsible for setting up the execution environment for the application + paradynd (transfer of all required files and start both programs).
- Currently, management of stdin, stdout and stderr is carried out according to Condor mechanisms.
- Once started, the application is controlled by the Paradyn Front-End through the paradynd, as usual.
- Termination of either the application or the paradynd implies the termination of the whole job (cleaning up the remote site).
Submitting a monitored job

Typical job description

universe = Vanilla
executable = foo
input = infile
output = outfile
arguments = 1 2 3
transfer_files = always

queue
Submitting a monitored job

Typical job description

- `universe = Vanilla`
- `executable = foo`
- `input = infile`
- `output = outfile`
- `arguments = 1 2 3`
- `transfer_files = always`
- `+SuspendJobAtExec = True`
- `+ToolDaemonCmd = "paradynd"`
- `+ToolDaemonArgs = "-zunix -l3 -mpinguino.cs.wisc.edu -p2090 -P2091 -a%pid"`
- `+ToolDaemonOutput = "daemon.out"`
- `+ToolDaemonError = "daemon.err"`
- `transfer_input_files = paradynd`
- `queue`
Starting a monitored job (Condor role)

- starter
  - createPausedProcess

Appl.
main()
Starting a monitored job (Condor role)

- starter
  - create_paused_process
  - create_normal_process
- paradynd
- monitored condor job
- Appl.
  - main()
Starting a monitored job (Paradyn role)

- **Starter**
  - create_paused_process
  - create_normal_process

- **Paradynd**
  - attach_to_created&paused

- **Monitored Condor Job**

- **Application**
  - main()
Starting a monitored job (Paradyn role)

Diagram:

- **starter**
  - `create_paused_process`
  - `create_normal_process`

- **paradynd**
  - `attach_to_created&paused`
  - `continue_proc`

- **monitored condor job**
  - **Appl.**
    - `main()`
    - `libc`

Actions:
- `stop`
Starting a monitored job (Paradyn role)
TDP Prototype

- **tool daemon**
  - `tdp_attach_to_paused`
  - `tdp_continue_proc`
  - `tdp library`

- **rpm**
  - `tdp_create_paused_process`
  - `tdp_create_normal_process`
  - `tdp library`

- **Appl.**
Problems and Side effects

• Resuming the execution of a process created by the starter and later controlled by the paradynd seems easy, BUT IT IS NOT!

• Two race conditions occur inside the Linux kernel with the scheduling of the application and the paradynd.

• An initial solution was found by sleeping the paradynd for a while and adding some extra synchronization (talk to Vic Zandy for details)

• A permanent solution requires changes in the Linux kernel.
Problems and Side effects
The termination problem in Linux

The starter has to be aware of local job’s termination

Process termination is normally detected using the signal mechanism.
Problems and Side effects

The termination problem in Linux (2)

In Linux, a monitoring process attached to an application becomes its parent.

Application termination is notified to the paradynd, but NOT to the starter...
Problems and Side effects

The termination problem in Linux (3)

The original parental relationship is recovered ONLY IF the monitor (i.e. the paradynd) calls wait()

Then, a second notification is sent to the original parent (i.e. the starter)

Condor has to trust the monitoring daemon or has to carry out periodic checks to test the state of the application
Current status

• A first prototype has been implemented in a Linux platform that provides the same basic functionality of the Paradyn “create process” mode.

• Vanilla jobs and MPI jobs are currently supported.

• Basic set of TDP functions are used to
  - create a paused process (included in Condor starter),
  - attach to and continue it (included in paradynd)
Work in Progress

- Add the ability to monitor an already running Condor job on the fly (i.e. Paradyn “attach” mode)

- Add new functionality to the TDP library that supports a normalized exchange of information between daemons
  - starter <-> paradynd
  - Front-End <-> schedd
  - What information is required? how it is exchanged?

- Adding support for other universes.
  - Standard jobs ⇒ challenges related to checkpointing
... and hopefully, see you in the next *Parador* Week