

A Portable Debugger for PVM / MPI Programs on IA64 Cluster

Weimin Zheng

Tsinghua University

Beijing China

What to talk about ?

- Introduction to Parallel Debugger
- Design of BUSTER
- Implementation of BUSTER
- Comparison with related work
- Feature Work

Introduction to Parallel Debugger

- Problems in debugging parallel programs
- Difficulties in using sequential debugger
- Main characters needed in parallel debugger

Introduction to Parallel Debugger

- Problems in debugging parallel programs
 - We must deal with synchronization problems when using message passing model.
 - Such problems may not occur in each execution. They depend on random factors, such as latency in communication. That is uncertainty problem in parallel debugging.

Introduction to Parallel Debugger

- Difficulties in using sequential debugger
 - It's difficult to navigate processes.
 - It's difficult to control processes.
 - It can not deal with uncertainty problems.

Introduction to Parallel Debugger

- Main characters needed in parallel debugger
 - Practicability.
 - It must support easy process navigation and control.
 - Scalability.
 - The number of processes being debugged should not be limited and can vary in runtime.
 - Portability.
 - A parallel debugger should support various platform used in high performance computing.

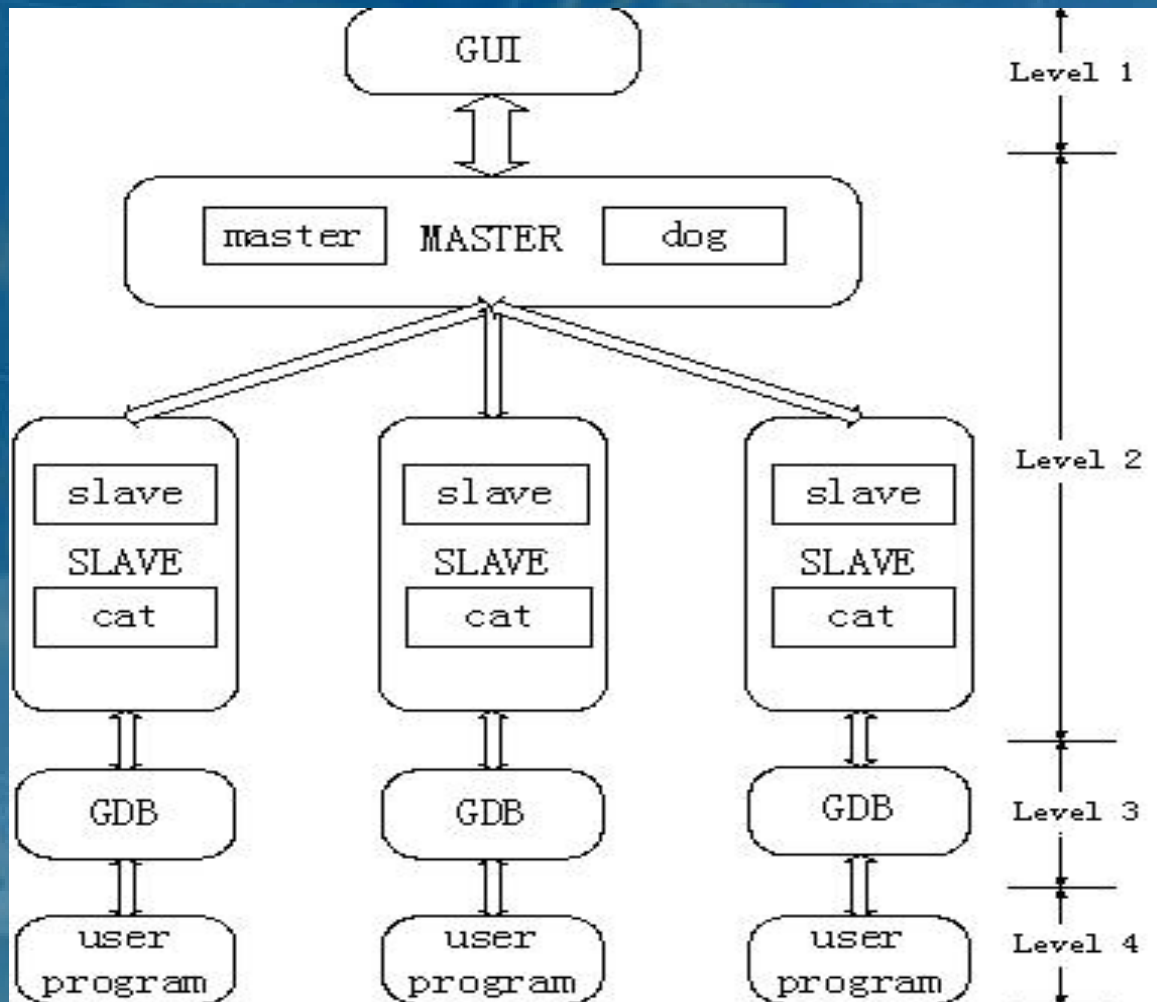
Design of BUSTER

- Two ways to build a parallel Debugger
- Four levels of design
- Data flow of BUSTER

Design of BUSTER

- Two ways to build a parallel debugger
 - Common model: Master-Slave
 - Master controls all slaves, and each slave controls one sequential debugger.
 - Using an existed sequential debugger.
 - Such as P2D2, Mantis, Xmdb and BUSTER.
 - More popular and portable. Performance depends on existed tools.
 - Using own developed sequential debugger.
 - Such as CM-5's Node Prism, TotalView and Intel's IPD.
 - More difficult to develop and port. Performance may be improved.

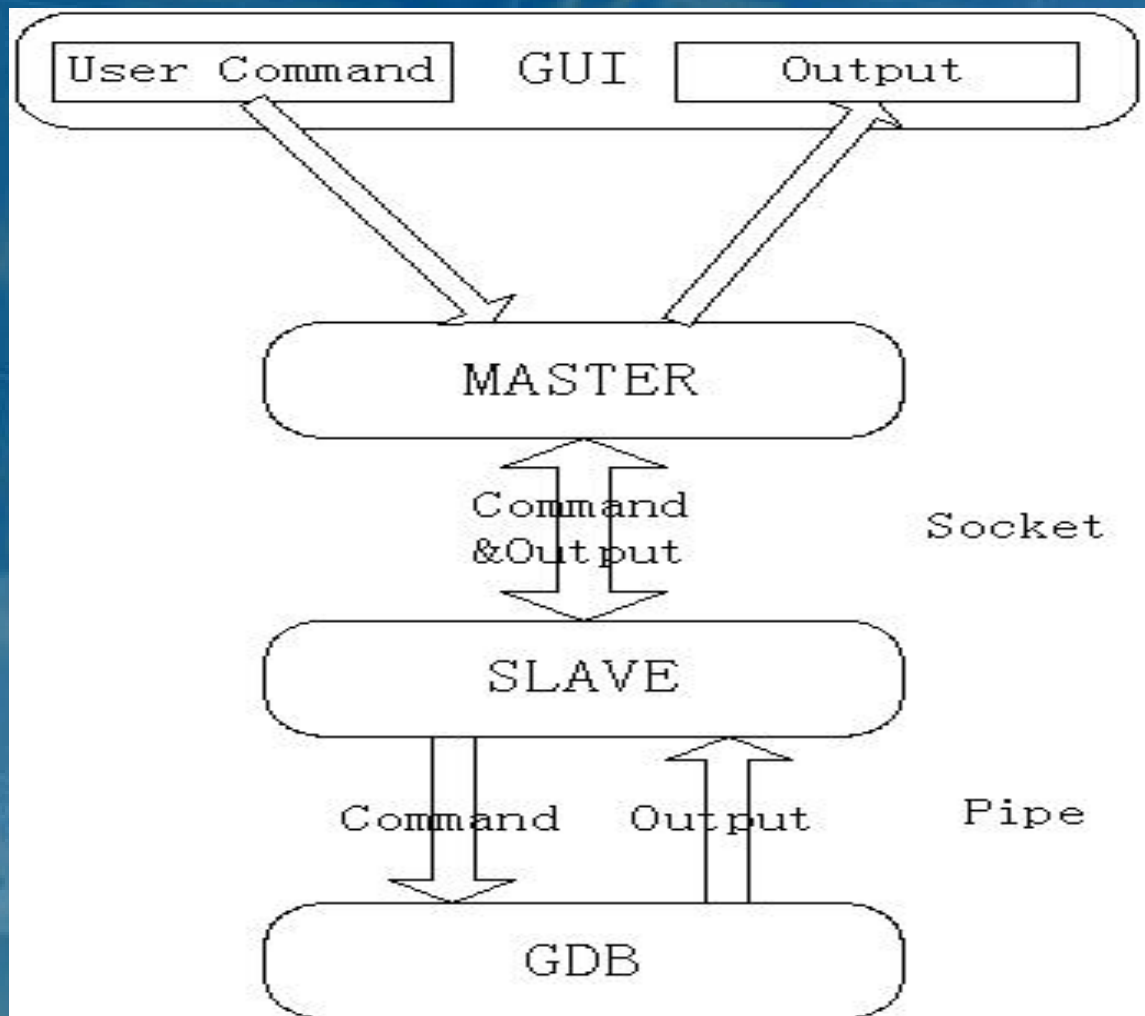
Four Levels of Design



Design of BUSTER

- Four levels of design
 - GUI Level
 - With a friendly graphic user interface. Receives users command and shows the result.
 - Control and Communication Level
 - Mainly divided into two module group: master and slave.
 - Sequential debugger level.
 - Execute commands forward by slave.
 - User's Program

Data Flow of BUSTER



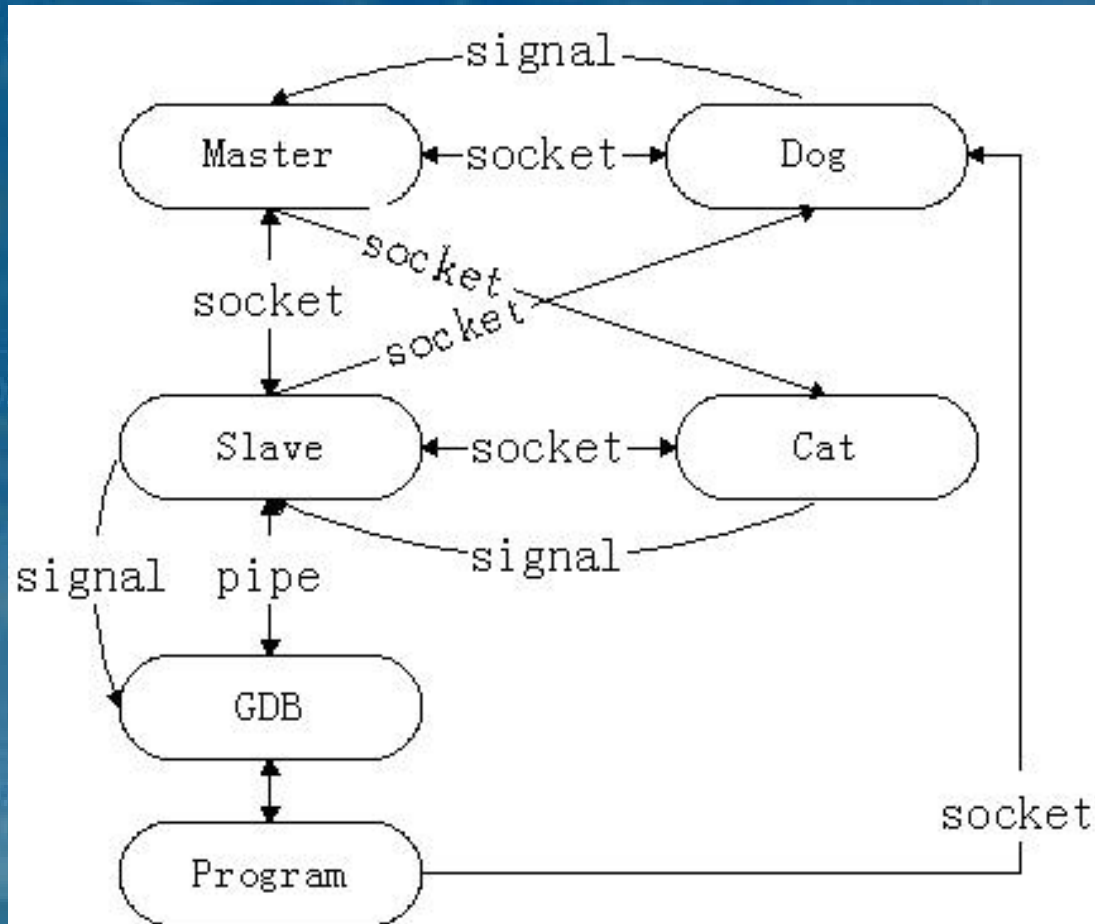
Design of BUSTER

- Data flow of BUSTER
 - Data flow is also divided by levels.
 - Both master and slave have two neighbors to communicate. That need precise protocol to prevent conflict. (Described later in implementation)

Implementation of BUSTER

- Communication protocol
- Automatic process detection
- Support for PVM and MPI
- Main characters achieved by design and implementation

Communication Protocol



Implementation of BUSTER

- Communication protocol
 - Synchronous command
 - Communication between master and slave directly. Master send command to slave and get ack from it.
 - Asynchronous command
 - Avoid conflict in communication.
 - Communication between master and slave using dog or cat.
 - Example: Master send a command to slave, then it keep busy in interacting with GUI. Slave execute the command and send ack to dog. Then dog signal master to get the ack.

Implementation of BUSTER

- Automatic process detection
 - Process spawned by some certain functions can be detected. Such as `pvm_spawn()` or `MPI_Init()`.
 - Implemented by changing the behavior of these functions.
 - Each spawn action actually spawn a new slave, then the slave will report to master, execute target program and control it.

Implementation of BUSTER

- Support for PVM and MPI
 - PVM: Modify user's source code
 - Include our head file in user's source file to replace pvm_spawn with our function.
 - MPI: Modify MPI library
 - MPI_Init use remote shell to execute target program in ch_p4 library. We make it execute our slave with all its parameters, so that slave can execute target program correctly.

Implementation of BUSTER

- Main characters achieved by design and implementation
 - Portability
 - By using level model and existed sequential debugger.
 - Scalability
 - By using automatic process detection
 - Practicality
 - By using fine GUI and GDB similar command.
 - By easy process navigation and control.
 - Robustness
 - By precise communication protocol.

Comparison with related work

- Mantis:
 - only supports debugging Split-C programs.
- DCDB:
 - It is more portable. But its way to control slave may make the load of local machine huge. BUSTER just uses two processes on local machine to control all the processes.
- TotalView:
 - It is a powerful commercial tools. It is only available on homogenous computer systems. Buster can support heterogeneous easily.
- Prism:
 - It is a data parallel debugger, neither portable nor general-purpose. Buster can run at most platform including IA64 cluster. Our design model is more general.

Feature Work

- Execution tracing and logging
 - Record message passing and other events in execution.
 - Message logged by modifying library; Other event logged by modifying user's source code.
- Global predicate detection
 - Determine potential problems by analyze execution log.
 - Enable global breakpoint function

Thanks To All !