MW: A framework to support Master Worker Applications

Sanjeev R. Kulkarni
Computer Sciences Department
University of Wisconsin-Madison
sanjeevk@cs.wisc.edu

Condor

Outline of the talk

- Introduction to MW
- Architecture of MW
- How to use MW?
- Records shattered by MW!
- Extensions

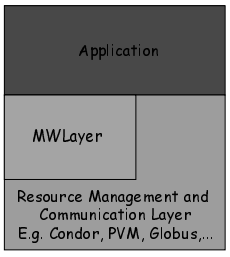
What is MW?

- MW = Master Worker
- Object Oriented, Fault Tolerant framework for Master-Worker Applications
- Can run on a variety of resource managers like Condor, PVM, Globus, ...

MW

- Object Oriented
  - MW is a set of C++ base classes
  - Users write a few virtual functions
- Fault Tolerant
  - Handles workers joining/leaving
  - Handles suspended/resumed workers
  - Checkpointing

MW Framework



Why use MW?

- Handles communication layer
- Handles resource Management
  - Useful especially in an opportunistic environment like Condor
- Same application code can run on various resource managers

MW Layer

- Three main entities
  - MWDriver
    - corresponds to the (Tyrant) Master
  - MWWorker
    - corresponds to the (exploited) Worker
  - MWTask
    - The work itself!

MWDriver: The Master

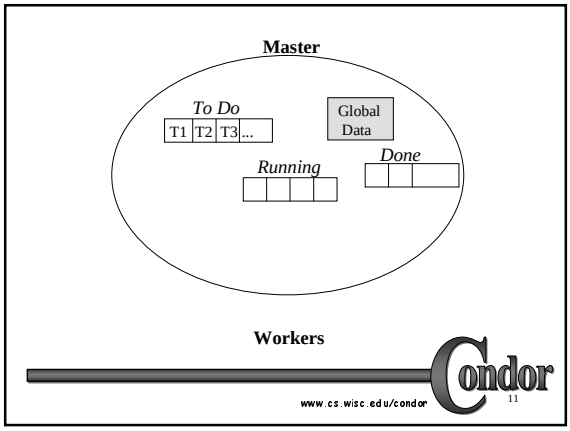
- Setup
- Manages workers joining/leaving
- Deals with HostSuspend/HostDelete
- Maintains lists of tasks
  - ToDo, Done, Running
- Acts on completed tasks
- Checkpointing

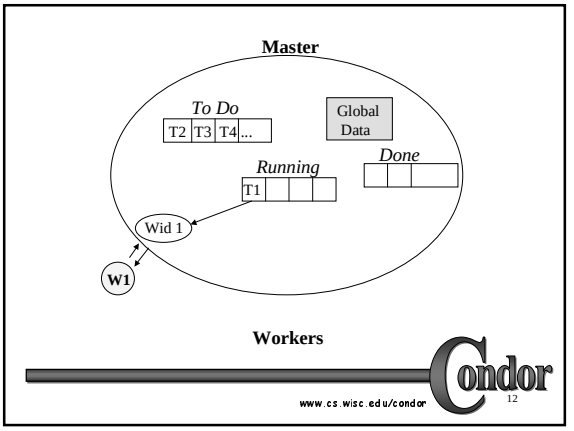
MWWorker: The Worker

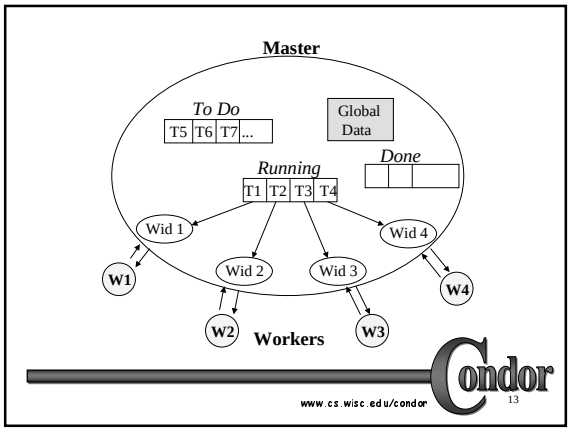
- Executes the given task
  - Unpacks the task got from Master
  - Executes the task
  - Packs results and sends back to Master

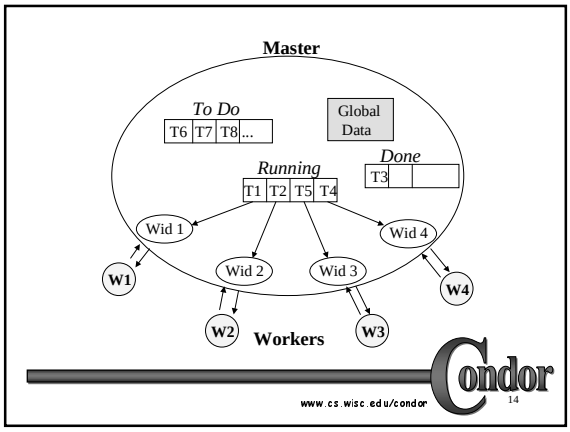
MWTask: The Work

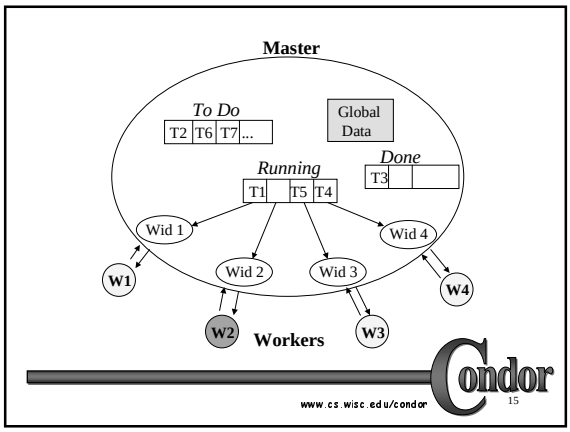
- Definition of a Unit of work
- Holds work to be done and results
- Follows the path *Master-Worker-Master*











Intelligent Scheduling of Tasks

- Some tasks may be harder
- Some machines may be faster
- Task ordering based on user defined MWKey
- Machine ordering based on Resource Manager information
  - e.g. Condor ClassAds

Checkpointing

- Save master state in case of failure
- Automatic restart from checkpoint file in case of master failure
- User controlled checkpoint frequency
- Users need to implement two additional functions to take advantage of these.

How can you use MW?

- MWDriver's virtual functions
  - *get_user_info*() - Processes arguments and does basic setup
  - *setup_initial_tasks*() - Fills in the ToDo list
  - *pack_worker_init_data*() - packs init data for worker
  - *act_on_completed_tasks*() - called every time a task completes. Optional.

Using MW (cont.)

- MWWorker
  - *unpack_init_data*() - unpack the init data sent by the master
  - *execute_task*() - execute a task and fill in the results
- MWTask
  - *pack_work*(), *unpack_work*()
  - *pack_results*(), *unpack_results*()

Resource Management and Communication Layer

- Presently two implementations exist
  - MW-PVM
    - Communication :PVM
    - Resource Manager : Condor-PVM
  - MW-File
    - Communication : Files
    - Resource Manager : Condor

MW-Independent

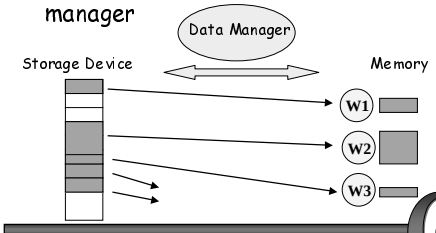
- Single process version
  - master
  - single worker
- sends and receives become `memcpy`
- No changes in source!
- Why?
  - Faster debugging

Data Management

- Exploitation of available memory as compared to CPU cycles
- Aims to reduce access to storage site by caching data on network
- Aimed at applications involving extremely large data sets

Data Manager

- Partitions large data sets into chunks
- Workers work on chunks allocated by manager

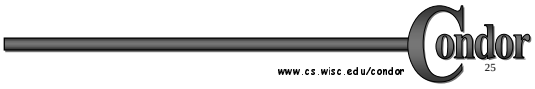


World Record Spree!!

- Stochastic Linear Programming
  - "Record breaking" problem of over 8.5M rows and 35M columns solved
- Quadratic Assignment Problem
  - nug27 in a little more than two days!
- Jeff will talk more in his talk

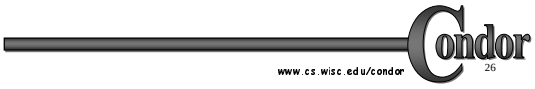
Extensions

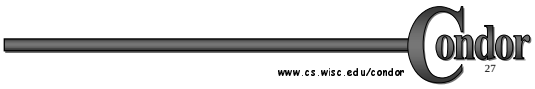
- More Resource Manager ports
  - Globus
  - A thread based version for SMPs
  - <put your suggestions here>



Scalability Issues

- Scales linearly with respect to the number of workers
- OK for 200 workers but for 1000?
- Solution
  - Hierarchical MW?



More about MW

- http://www.cs.wisc.edu/condor/mw
- Demos of MW in action using Condor pool *tomorrow in Room 3397*