A Master-Worker Framework for MetaComputing

JEAN-PIERRE GOUX
JEFF LINDEROTH

Argonne National Labs
Math and Computer Science Division
{goux,linderoth}@mcs.anl.gov

MIKE YODER
SANJEEV KULKARNI

University of Wisconsin
Department of Computer Science
{yoderme,sanjeevk}@cs.wisc.edu

Outline

- MWMotivation and MWHistory
- What you gain by using MW
- How to use MW
- Successful use of MW

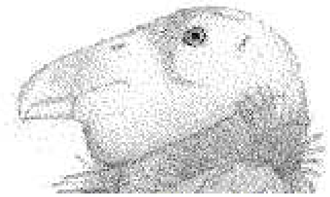
MWHistory – metaNEOS

- If we are to believe the grant proposal, metaNEOS should...

  "Turn existing networked computing resources into productive problem solving environments by integrating advanced distributed resource managers, novel optimization tools, blah blah blah"

- One such "networked computing resource" is a Condor cluster

What is Condor?

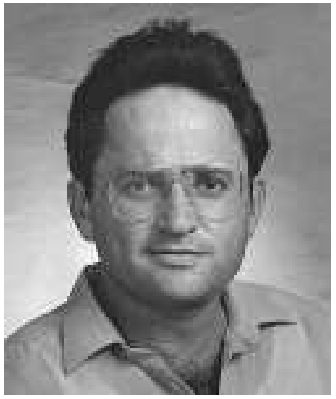


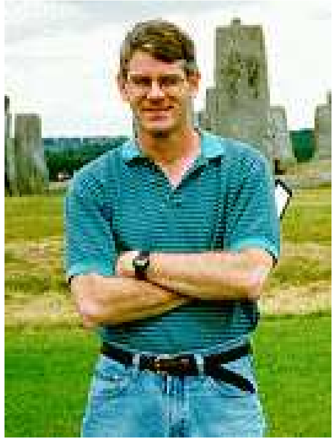
- Manages collections of “distributively owned” workstations
  - ◇ Workstation owner specifies conditions under which jobs are allowed to run
  - ◇ Jobs can be checkpointed and migrated
  - ◇ Remote system calls provide the “originating” machines environment

Metacomputers are...

A General Metacomputer	Our Metacomputer
Dynamic	*Very* Dynamic
Heterogenous	3 Architectures, 2 Operating Systems Speeds From 30 - 453 MIPS Two LP Solvers – CPLEX and SOPLEX
"Communicationally Challenged"	All machines live in Wisconsin (for now)
Large Scale	Over 350 machines in the pool, but we have to share them.

MWHistory – The Play...

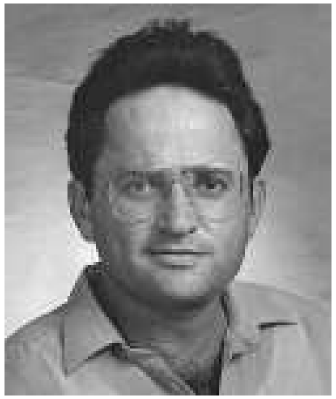


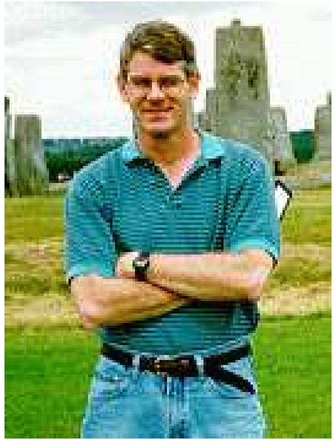


"Why can't you use Condor and batch submit your optimization jobs to run?"

- Optimization algorithms are iterative
- Initializing a process may require sending a large amount of startup data and startup computations
- Communication needs are likely to be message passing – not file-based

WoDi to the rescue?!?!





- Use WoDi
  - ◇ Intended to assist in writing "master-worker" applications for the Condor dynamic environment
  - ◇ Work must be performed in cycles

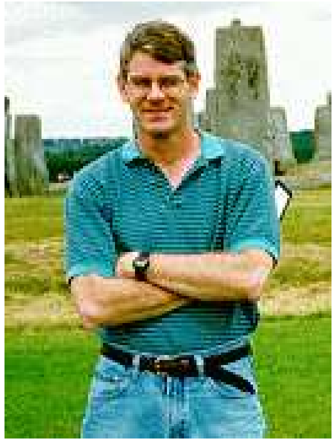
- Optimization algorithms don't necessarily have "work cycles"

⋆ The author of WoDi was long gone!!

Our hero!!!

- Mike Yoder wrote a package called SlaveDriver that used Condor/PVM to perform master-worker computations in the Condor environment
- Condor-PVM – A modified PVM that interfaces with Condor. The user is notified through a message of the following events:
  - HostAdd – A machine is available for your use
  - HostSuspend – Someone touched the keyboard of a machine
  - HostDelete – Your machine has gone away
- The user is responsible for taking appropriate actions in the case of these things happening

Using SlaveDriver



- "I'll take Mike's SlaveDriver code and implement a cutting plane code for stochastic programming"
- Days Pass... Steve must finishing writing epic novel *Numerical Optimization*
- "Jeff, why don't you take the SlaveDriver code and implement a cutting plane code for stochastic programming"

Evolving SlaveDriver

- Truly a discrete Optimizer at heart, Jeff *really* wanted to implement a branch and bound code, and realized that with minor modifications, SlaveDriver could do both!!!

- Jean-Pierre, Mike, and Jeff set out abstract the SlaveDriver code so that it could execute arbitrary master-worker applications

- Ever the champions of political correctness, they changed the name to MW

From the home office in Wisconsin Rapids, Wisconsin...

Top Eight MWAcronyms

from Mike Yoder
Jean-Pierre Goux
Jeff Linderoth
and `grep /usr/dict/words`

8 Master-Worker

7 Metaneos-Wisconsin

6 MeltdoWn

5 Muy-Wicked

4 Mostly Wasted

3 Meritorius Whirligig

2 Maladroit Wastrel

And the number one MWAcronym is....

1 Mike's Work!!!

MWHistory – The Play's Exciting Conclusion

- MW – A number of abstract classes from which users derive funcitonality to solve scientific computing problems
- Hopefully, it is...
  - Easy to use
  - Reliable and Adaptable
  - Efficient

What MW does for you

- ⋆ **LOTS** of "free" computing cycles
- Transparently adapts to dynamic environment
  - ◇ Fault-tolerance and new processors joining in
  - ◇ Spawns proper executables depending on machine OS and arbitrary "required" features
  - ◇ Data Conversion Thank you PVM!!!
- Will do user-defined checkpointing and restart from a checkpoint
  - ◇ User must write functions to read and write "state"
  - ◇ Condor-PVM does *not* allow checkpointing of individual tasks. But other systems built with MW do!!!

But Wait, There's More!

- Statistics
  - ◇ Keeps track of each workers running, idle, and suspended times, and generates "efficiency" statistics.
  - ◇ (Hopefully soon) Automatic SLOG file generation
- Users can set host suspension and deletion policies
- Hooks for the user to manage the list of tasks in case all tasks are not created equal
- (Hopefully soon) "Intelligent" task scheduling policies
- (Hopefully soon) User-defined benchmarking capabilities

SOLD!!!!

- To use MW for an application, you must implement functionality derived from three abstract base classes:
  - MWDriver
  - MWTask
  - MWWorker

MWApplications

- MWFATCOP – A branch and cut code for linear integer programming
- MWMINLP – A branch and bound code for nonlinear integer programming
- MWSVM – A column generation code for solving support vector machine problems
- MWLShaped – A cutting plane code for linear stochastic programming
- MWQAP – A branch and bound code for solving the quadratic assignment problem
- . . . (Your application here) . . .

Stochastic Programming – In-Depth Description

- Given a (linear) Optimization problem with a number of scenarios K that might occur at a later time

1. Solve "master" linear program to get guess on optimal expected value solution $\hat{x}$
2. Solve K subproblem linear programs to see if the guess on $\hat{x}$ was right.

Yes? YAHOO!

No? Add "cut(s)" to the master linear program. Goto 1.

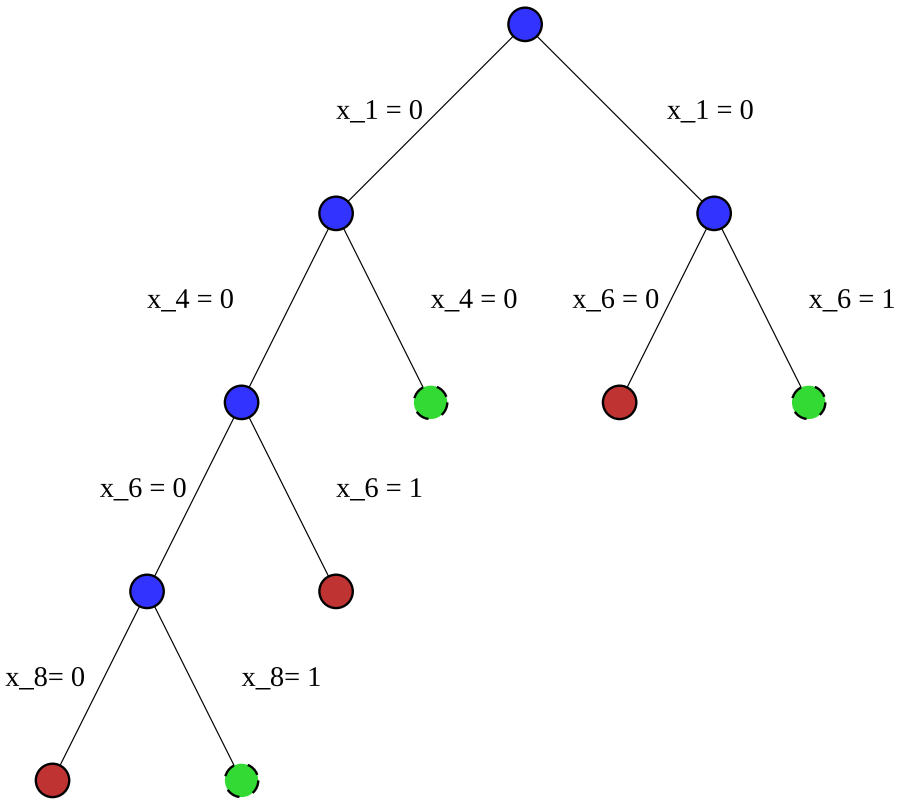
Branch and Bound – In-Depth Description

- Feasible solution ⇒ upper bound
- Relaxed problem ⇒ lower bound

1. Is solution to relaxed problem feasible?

Yes? YAHOO!

No? Break problem into smaller pieces. Goto 1.



MWDriver

- Handles all messages regarding states of workers, and responsible for assigning MWTasks to MWWorkers
- User must implement
  - `get_userinfo()` – Reads data and performs problem-specific initializations
  - `setup_initial_tasks()` – Adds the initial units of work to the task pool
  - `pack_worker_init_data()` – Packs initial data that is sent to workers upon initialization
  - `act_on_completed_task()` – Performs appropriate actions on completion of an MWTask

Drivers for Optimization Algorithms

	MWQAP	MWLShaped
`pack_worker_init_data()`	All information about relaxed problem	All scenario information
`act_on_completed_task()`	Add unsolved nodes to the work pool	If all K subproblems have been solved, solve master LP with new cut(s) and generate new $\hat{x}$

MWTask

- The definition of one unit of work.
  - Contains both the work to be done and the result of the work.
- 4 functions to implement
  - `(un)pack_work`
  - `(un)pack_result`

Tasks for Optimization Algorithms

	MWQAP	MWLShaped
Work	• One node of the search tree • An entire subtree of the search tree	• $\hat{x}$ • Beginning index b • Ending index e
Result	• Unfinished nodes • Improved solution value	• A cut (or no cut)

MWWorker

- User must implement...
  - `unpack_worker_init_data()`
    - Unpacks initialization data
    - Sets up worker to receive MWTasks
  - `execute_task()`
    - Fills in the result portion of the MWTask
    - Usually where the meat of the scientific code resides

Workers for Optimization Algorithms

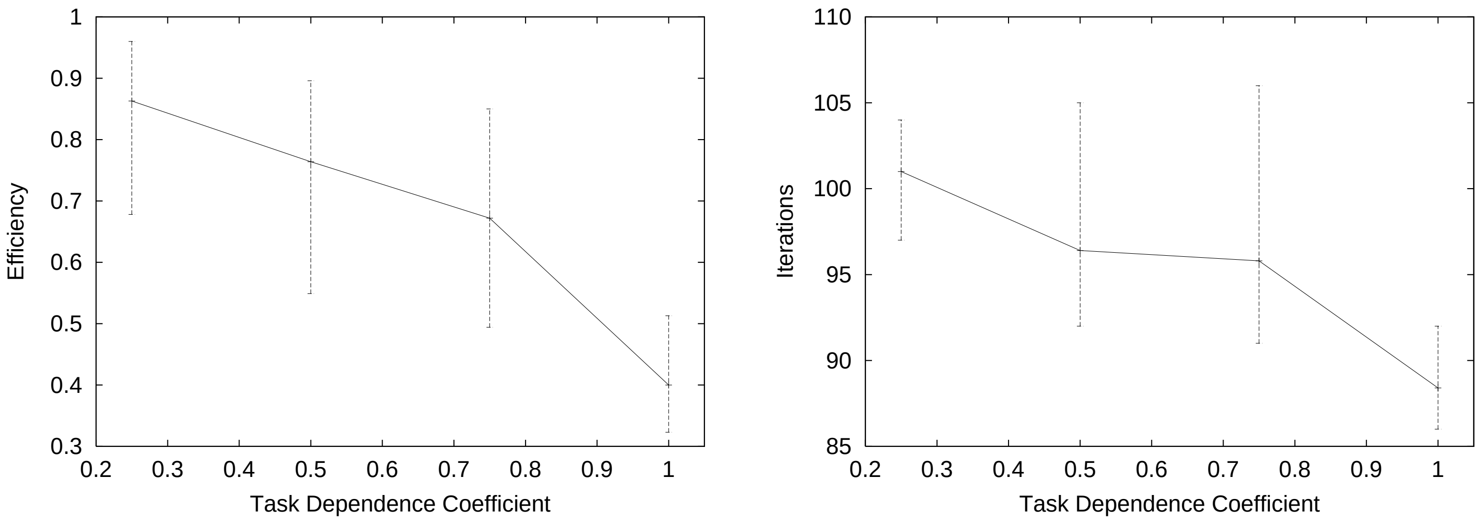
	MWQAP	MWLShaped
`execute_task()`	Evaluate node (or subtree)	Solve all subproblem LP's whose index in K is between b and e

MWUseless

- The master-worker paradigm is not scalable, so how can can we make MW do useful things?
- Thankfully, these optimization algorithms possess (or have been made to possess) one or both of the following key features:
  - **Dynamic Grain Size**. The overall computation can be easily broken into pieces of work of variable size.
  - **Weak synchronization**. The ability to execute a task does not depend on the completion of a large number of other tasks.
- Have very little experience at trying to scale to $>$ 100 processors
- The point is that we can reliably deliver these processors for long time spans

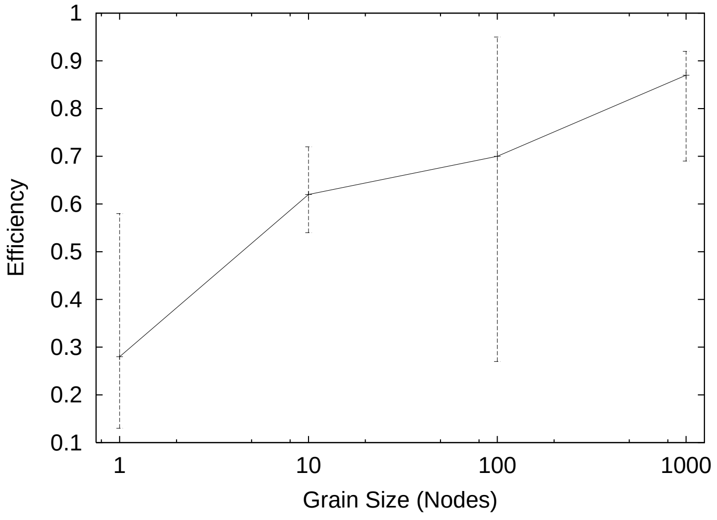
Obligatory Figures – 1

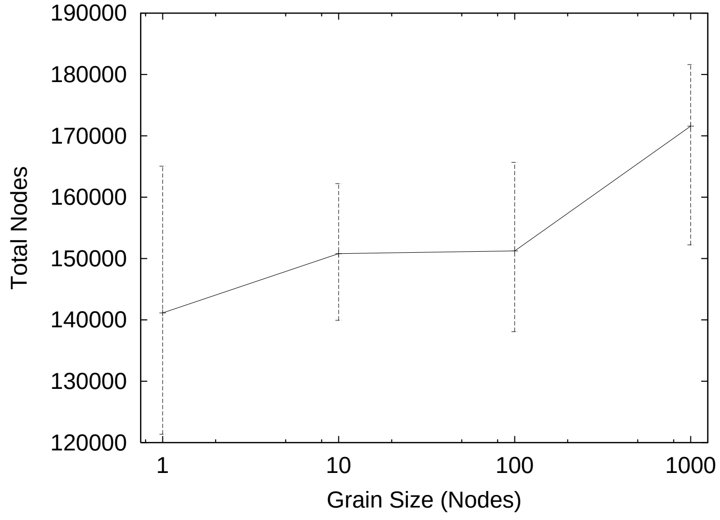
- MWLShaped – The effect of synchronicity on efficiency and iteration count



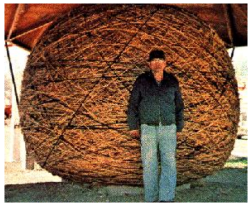
Obligatory Figures – 2

- MWQAP – The effect of grain size on efficiency and node count





World Records!



- MWMINLP
  - Solved previously unsolved instances of a trim-loss problem
- MWQAP
  - Have solved difficult instances in a fraction of the time (> 3 months $\Rightarrow$ < 1 day)
  - Will "soon" solve an instance that has been open for > 35 years (10^5 CPU Hours)
- MW/FATCOP
  - Will (probably "soon") solve the most famous open integer programming problem

The MWFuture

- Nice front-end to MW applications which gives complete "problem solving environment" – iMW
- Other communication and resource management layers
  - MWFiles – A file based messaging passing mechanism making use of Condor's remote I/O
  - Globus-based tools

The MWFuture

- Ways to make MW more "data-aware"
  - All algorithms are task parallel!
  - Can data parallel algorithms be abstracted nicely in this framework
- Dealing with TRULY large numbers of resources
  - First, we must *get* a large number of resources. Chiba City?
  - Will likely require something like a heirarchy of masters, or a controlling master
  - The user's underlying application will remain (largely?) unchanged

`http://www.cs.wisc.edu/condor/mw`