MW: A Master-Worker Toolkit for Implementing Operations Research Algorithms on the Computational Grid

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

Fourth International Workshop of the EURO
Working Group on Parallel Processing in Operations Research

Mont-Tremblant (Quebec), Canada   January 17, 2005

MWCollaborators

- Greg Thain, Wen-Han Goh, Sanjeev Kulkarni, Miron Livny, Steve Wright, Mike Yoder
  - University of Wisconsin-Madison
- Jean-Pierre Goux
  - Northwestern University and Argonne National Lab
- Kurt Anstreicher, Nate Brixius
  - University of Iowa

Outline

- MWMotivation
  - MWHistory
  - Computational Grids
- MWDesign
  - The MW API
  - The MW IPI (Infrastructure Programming Interface)
- MWSuccesses
  - Stochastic Linear Programming
  - The Quadratic Assignment Problem—Solving nug30.

MWMotivation

- 1998, metaNEOS—Metacomputing Environments for Optimization.
    - NSF Grant to explore solving large scale numerical optimization problems on large scale computing platforms
- Our aim was to show-off by solving BIG problems
    - Better algorithms?
    - More powerful computers!
- How about supercomputers?
    - e.g. NCSA — National Center for Supercomputing Applications.
    - Nearly 2000 processor "super-cluster"

Problems With Using Supercomputers

A Look At The Job Queue

```
[linderot@tunb ~] bqueues
QUEUE_NAME      PRIO      MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
cap1             60        -    -    -    -   626   114   498     0
cap2             60        -    -    -    -   784   560   224     0
cap3             60        -    -    -    -   160     0   160     0
ind              60        -    -    -    -     0     0     0     0
debug            50        -    -    -    -     0     0     0     0
normal           30        -    -    -    - 10819  9691  1054     0
admin            30        -    -    -    -     0     0     0     0
```

- Over 10,000 pending jobs!
- You can queue your job and wait (literally) days until it will run.

On a Positive Note

How Many Processors Are Available?

```
[linderot@tunb ~] bhosts | grep 'ok' | wc -l
    231
```

- There are 231 processors that are currently available!
- They are being "saved" to run the next big parallel job in the queue.
- "Backfill": We could use those processors for our computation, but we have to schedule them for a short time period
- Use the processors as part of a larger computation
    - We must be able to handle processors "going away"

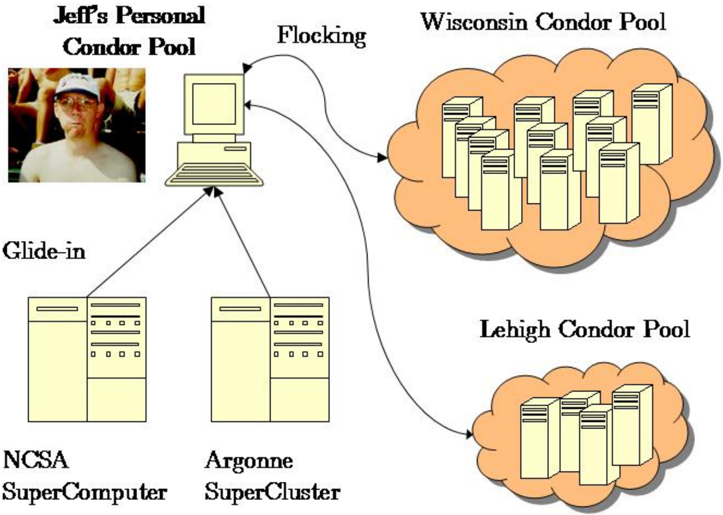
Getting Even More Computers

- I might be able to use "your" computer too
  - If I only used it while you weren't looking.
- People envision a "Computational Grid" made up of CPU cycles that would otherwise go wasted
  - Analogy is to the national power grid
  - CPU Cycles are a ubiquitous and nearly endless resource, if only you can harness them
- To create and use a Grid, we need software tools to
  1. Locate and harness CPU cycles
  2. Manage these CPU cycles for our parallel algorithms

Building Grids with Condor

- Manages collections of "distributively owned" workstations
  - User need not have an account or access to the machine
  - Workstation owner specifies conditions under which jobs are allowed to run—Jobs must vacate when user claims machine!
- How does it do this?
  - Scheduling/Matchmaking
  - Jobs can be checkpointed and migrated
  - Remote system calls provide the originating machines environment
- Flocking: Jobs in one "Condor Pool" can negotiate to run in other Condor pools
- Glide-in: Nodes can "temporarily" join an existing Condor pool.

Personal Condor



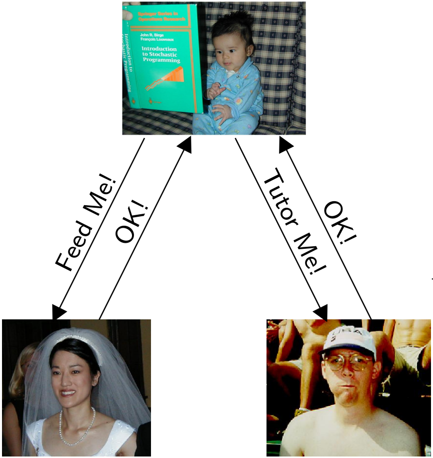
Grid-Enabling Algorithms

- Condor and "glide-in" gives us the infrastructure from which to build a grid (the spare CPU cycles),
- We still need a mechanism for controlling the algorithm on a computational grid
- No guarantee about how long a processor will be available.
- No guarantee about when new processors will become available

---

- To make parallel algorithms dynamically adjustable and fault-tolerant, we could (should?) use the master-worker paradigm
- What is the master-worker paradigm, you ask?

Master-Worker!



- Master assigns tasks to the workers
- Workers perform tasks, and report results back to master
- Workers do not communicate (except through the master)

- Simple!
- Dynamic/Fault-tolerant
- Reusable(!?)

MW : A Master-Worker Grid Toolkit

- There are three abstraction in the master-worker paradigm: Master, Worker, and Task.
- MW is a software package that encapsulates these abstractions
  - API : C++ abstract classes
  - User writes 10 methods
  - The MWized code will transparently adapt to the dynamic and heterogeneous computing environment
- MW also has abstract layer to resource management and communications packages (an Infrastructure Programming Interface).
  - Condor/PVM
  - Condor/Files
  - Static/MPI
  - Single processor

MW API

- MWMaster
  - get_userinfo()
  - setup_initial_tasks()
  - pack_worker_init_data()
  - act_on_completed_task()
- MWTask
  - pack_work(), unpack_work()
  - pack_result(), unpack_result()
- MWWorker
  - unpack_worker_init_data()
  - execute_task()

MW IPI

- MW can be used with other grid toolkits as long as the following functionality can be provided:
- Communication
  - `pack()`, `unpack()`, `send()`, `recv()`
  - Message buffer management routines
  - Changes in machine state are passed to master as tagged messages (HOSTADD, HOSTDELETE, etc.)
- Resource Management
  - `set_target_num_workers(int num_workers)`
  - `get_worker_info(MWWorkerID *)` : `MWWorkerID` class has members such as architecture, operating system, machine speed, etc.
  - `start_worker( MWWorkerID * )`

Optimization Algorithms...

- Are iterative
  - Generally not "pleasantly parallel"
- Use data
  - Incrementally
  - "Optionally" (Potentially computed instead of shared)
- Are weakly synchronous
  - Can have their sychronization requirements reduced at a modest performance penalty
- Have a dynamic grain size
  - The computation can "easily" be broken into pieces of variable size.

A Key Idea!

Exploit these features to fit the algorithms onto a Computational Grid computing platform.

MW Applications

- MWFATCOP (Chen, Ferris, Linderoth) – A branch and cut code for linear integer programming
- MWMINLP (Goux, Leyffer, Nocedal) – A branch and bound code for nonlinear integer programming
- MWQPBB (Linderoth) – A (simplicial) branch and bound code for solving quadratically constrained quadratic programs
- MWAND (Linderoth, Shen) – A nested decomposition based solver for multistage stochastic linear programming
- MWATR (Linderoth, Shapiro, Wright) – A trust-region-enhanced cutting plane code for linear stochastic programming and statistical verification of solution quality.
- MWQAP (Anstreicher, Brixius, Goux, Linderoth) – A branch and bound code for solving the quadratic assignment problem

Stochastic Programming

A Stochastic Program

$$\min_{x \in X} f(x) \stackrel{\text{def}}{=} \mathbb{E}_\omega F(x, \omega) \qquad \text{(SP)}$$

- Typically, we must make decision x before $\omega \in \Omega$ is known

---

1. We make a decision now (first-period decision)
2. Nature makes a random decision ("stuff" happens)
3. We make a second period decision that attempts to repair the havoc wrought by nature in (2). (recourse)

- Multistage problems are defined by a sequence of decision, event, decision, event, . . ., decision.

Two-Stage Stochastic LP with Recourse

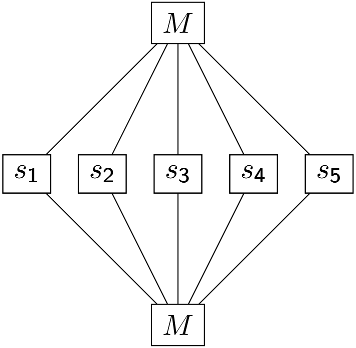
- Imagine the case where $\Omega = \{\omega_1, \omega_2, \ldots \omega_S\} \subseteq \Re^r$.
- $\mathsf{P}(\omega = \omega_s) = p_s, \forall s = 1, 2, \ldots, S$

$$\min_{x \in \mathbb{R}^n_+} \{c^T x + \mathcal{Q}(x) : Ax = b\}$$

$$\mathcal{Q}(x) \stackrel{\text{def}}{=} \sum_{s=1}^{S} p_s \left[\min_{y_s \in Y} \{q^T y_s : W y_s = h_s - T_s x\} \right]$$

- This is a (nonlinear) programming problem in $\Re^n$.
- $\mathcal{Q}(x)$ is...
  - Convex, Continuous, Non-differentiable
  - Evaluation of $\mathcal{Q}(x)$ also yields subgradients

Work-Cycle Computation



1. Solve the **master problem** M with the current θ_j-approximations to $\mathcal{Q}_{[j]}(x)$ for x^k.
2. Solve the **subproblems**, (s_j) evaluating $\mathcal{Q}_{[j]}(x^k)$ and obtaining a subgradient $g_j(x^k)$. Add inequalities to the master problem
3. k = k+1. Goto 1.

More Bells and Whistles

- Equip the L-Shaped method with a trust region
- Allow asychronous evaluation of a whole "basket" of iterates $\{x^{k_1}, x^{k_2}, \ldots, x^{k_B}\}$
- Show off by solving "The World's Largest Linear Program"
- Storm – A cargo flight scheduling problem (Mulvey and Ruszczyński)
- We aim to solve an instance with 10,000,000 scenarios
- $x \in \Re^{121}, y_s \in \Re^{1259}$
- The deterministic equivalent is of size

$$A \in \Re^{985{,}032{,}889 \times 12{,}590{,}000{,}121}$$

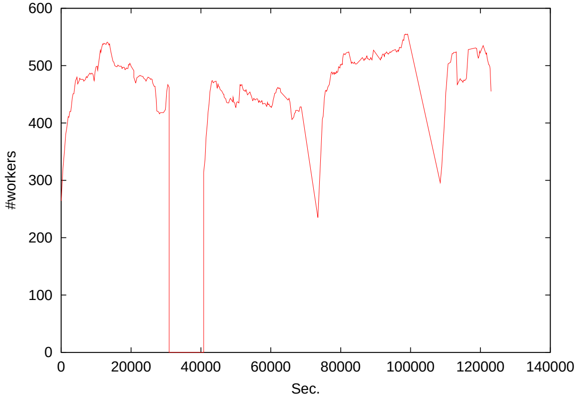
The Super Storm Computer

Number	Type	Location
184	Intel/Linux	Argonne
254	Intel/Linux	New Mexico
36	Intel/Linux	NCSA
265	Intel/Linux	Wisconsin
88	Intel/Solaris	Wisconsin
239	Sun/Solaris	Wisconsin
124	Intel/Linux	Georgia Tech
90	Intel/Solaris	Georgia Tech
13	Sun/Solaris	Georgia Tech
9	Intel/Linux	Columbia U.
10	Sun/Solaris	Columbia U.
33	Intel/Linux	Italy (INFN)
1345		

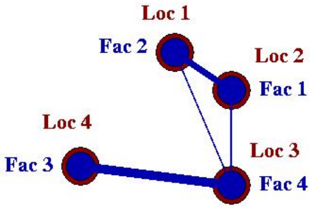
TA-DA!!!!!

Wall clock time	31:53:37
CPU time	1.03 Years
Avg. # machines	433
Max # machines	556
Parallel Efficiency	67%
Master iterations	199
CPU Time solving the master problem	1:54:37
Maximum number of rows in master problem	39647

Number of Workers



The Quadratic Assignment Problem

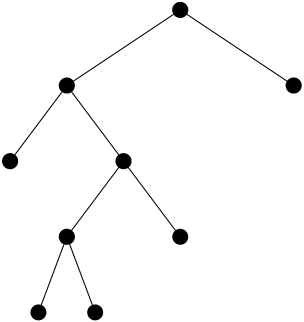


The Quadratic Assignment Problem

$$\min_{\pi} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i),\pi(j)} + \sum_{i=1}^{n} c_{i\pi(i)}$$

- Assign facilities to locations minimizing total distance flow between facilities must travel
- QAP is NP-Hard
  - Branch-and-bound is the method of choice

Tree-Based Computations



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

Branch-and-Bound

1. Is solution to relaxed problem feasible?

   Yes? YAHOO!

   No? Break problem into smaller pieces. Goto 1.

The Devil In The Details

- Fitting the B & B algorithm into the master-worker paradigm is not groundbreaking research
- We must avoid contention at the master
  - Reduce arrival rate : Have machines work on a task for a sufficiently long time (Dynamic Grain Size)
  - Increase service rate : Do *not* have workers pass back many nodes. Keep master's list of tasks small.
- Balancing efficiency considerations with search considerations was very important! (50% $\rightarrow$ 90%)!
- We contend that with appropriate tuning, *many* algorithms can be shoehorned into the master-worker paradigm!

MW can be a grid computing workhorse!

The Holy Grail

- nug30 (a QAP instance of size 30) had been the "holy grail" of computational QAP research for >30 years
- In 2000, Anstreicher, Brixius, Goux, & Linderoth set out to solve this problem
- Using a mathematically sophisticated and well-engineered algorithm, we still estimated that we would require 11 CPU years to solve the problem.

The nug30 Computational Grid

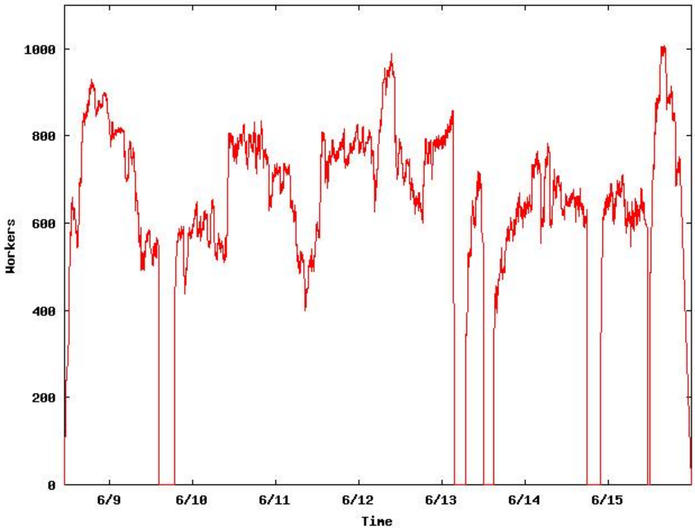
Number	Type	Location	How
96	SGI/Irix	Argonne	Glide-in
414	Intel/Linux	Argonne	Glide-in
1024	SGI/Irix	NCSA	Glide-in
16	Intel/Linux	NCSA	Flocked
45	SGI/Irix	NCSA	Flocked
246	Intel/Linux	Wisconsin	Flocked
146	Intel/Solaris	Wisconsin	Flocked
133	Sun/Solaris	Wisconsin	Flocked
190	Intel/Linux	Georgia Tech	Flocked
96	Intel/Solaris	Georgia Tech	Flocked
54	Intel/Linux	Italy (INFN)	Flocked
25	Intel/Linux	New Mexico	Flocked
12	Sun/Solaris	Northwestern	Flocked
5	Intel/Linux	Columbia U.	Flocked
10	Sun/Solaris	Columbia U.	Flocked
2510			

NUG30 is solved!

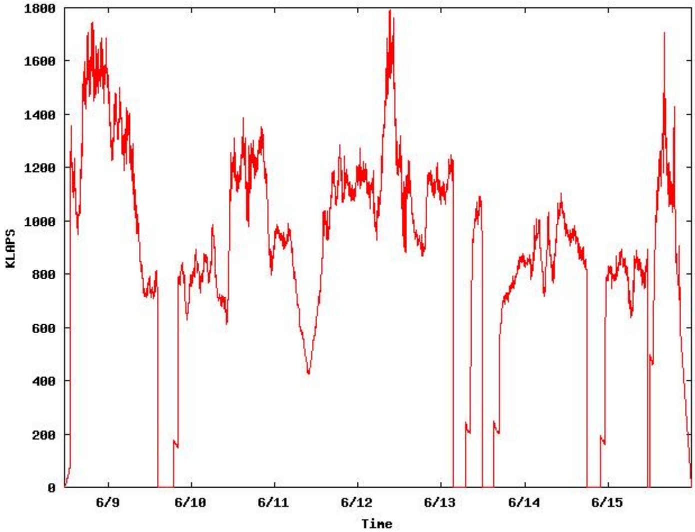
14, 5, 28, 24, 1, 3, 16, 15, 10, 9, 21, 2, 4, 29, 25, 22, 13, 26, 17, 30, 6, 20, 19, 8, 18, 7, 27, 12, 11, 23

Wall Clock Time:	6:22:04:31
Avg. # Machines:	653
CPU Time:	$\approx$ 11 years
Nodes:	11,892,208,412
LAPs:	574,254,156,532
Parallel Efficiency:	92%

Workers



KLAPS


1800
1600
1400
1200
1000
800
600
400
200
0
KLAPS
6/9
6/10
6/11
6/12
6/13
6/14
6/15
Time

Even More Wasted CPU Time

	KRA30B	KRA32	THO30
Wall Clock Time (Days)	3.79	12.3	17.2
Avg. # Machines	462	576	661
Max. # Machines	780	1079	1307
CPU Time (Years)	4.32	15.2	24.7
Nodes	5.14×10^9	16.7×10^9	34.3×10^9
LAPs	188×10^9	681×10^9	1.13×10^{12}
Parallel Efficiency:	92%	87%	89%

Conclusions

- The master-worker paradigm can be effectively used to distribute many parallel operations research applications
  - Maybe yours too!
- The master-worker paradigm is nicely suited to a Grid implementation
  - We really believe that master-worker is the "right" paradigm for distributed computing on the Grid
- MW can make implementing master-worker algorithms for the Grid easier

The End!

We want YOU to join the MW community of users

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

`mailto:jtl3@lehigh.edu`