MW: The Master-Worker Library

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

Paradyn/Condor Week
University of Wisconsin

Madison, WI

March 14, 2005

MWCollaborators

- Greg Thain
- Wen-Han Goh
- Sanjeev Kulkarni
- Miron Livny
- Steve Wright
- Mike Yoder
- Pete Keller
- Jichuan Chang
- Alan Bailey
- Minyi Xu

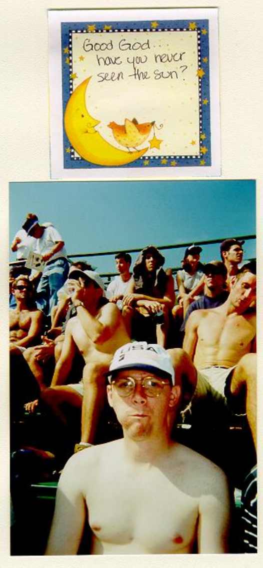
- Jean-Pierre Goux

Outline

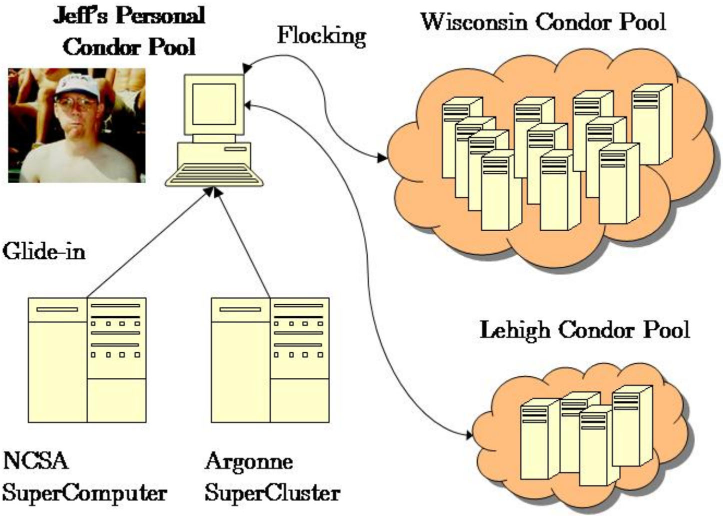
- MWMotivation
- MWDesign
- MWSuccesses
  - Stochastic Linear Programming
  - The Quadratic Assignment Problem—Solving nug30.
- MWFuture

Meet Jeff!

Jeff wants to solve large numerical optimization problems



Jeff's Personal Condor



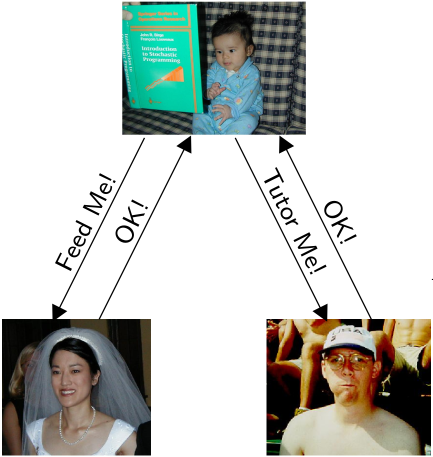
Grid-Enabling Algorithms

- Condor and "glide-in" give Jeff the infrastructure from which to build a grid (the spare CPU cycles),
- Jeff still needs a mechanism for controlling a (large) distributed 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, Jeff 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!
- Fault-tolerant
- Dynamic

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, Files, Socket}
  - 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()`

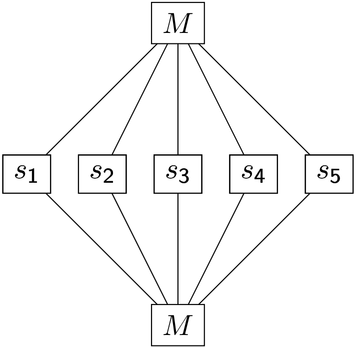
MW and Condor!

- Think of MW as a more dynamic and flexible DAG-Man
- It's also more complicated to use

MW Applications

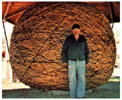
- MWFATCOP (Chen, Ferris, Linderoth) – A branch and cut code for linear integer programming
- MWMINLP (Goux, Leyffer, Nocedal) – A branch and bound code for nonlinear integer programming
- MWQPBB (Linderoth) – A (simplicial) branch and bound code for solving quadratically constrained quadratic programs
- 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 LP—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.

Show-Off!



- (with Steve Wright), Jeff aims to show off by solving "The World's Largest Linear Program"
- Storm – A cargo flight scheduling problem (Mulvey and Ruszczyński)
- Solve an instance with 10,000,000 scenarios
- $x \in \Re^{121}, y_s \in \Re^{1259}$
- The deterministic equivalent is of size

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

Jeff's Super Storm Computer

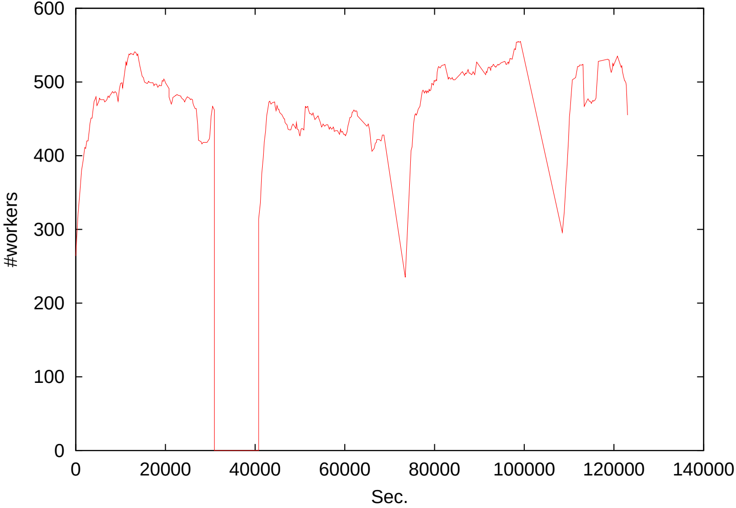
Number	Type	Location
184	Intel/Linux	Argonne
254	Intel/Linux	New Mexico
36	Intel/Linux	NCSA
265	Intel/Linux	Wisconsin
88	Intel/Solaris	Wisconsin
239	Sun/Solaris	Wisconsin
124	Intel/Linux	Georgia Tech
90	Intel/Solaris	Georgia Tech
13	Sun/Solaris	Georgia Tech
9	Intel/Linux	Columbia U.
10	Sun/Solaris	Columbia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

TA-DA!!!!!

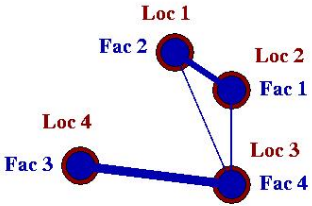
Computation Statistics

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

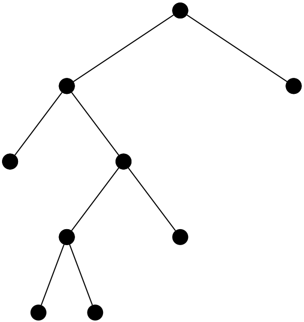


The Quadratic Assignment Problem

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

- 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			

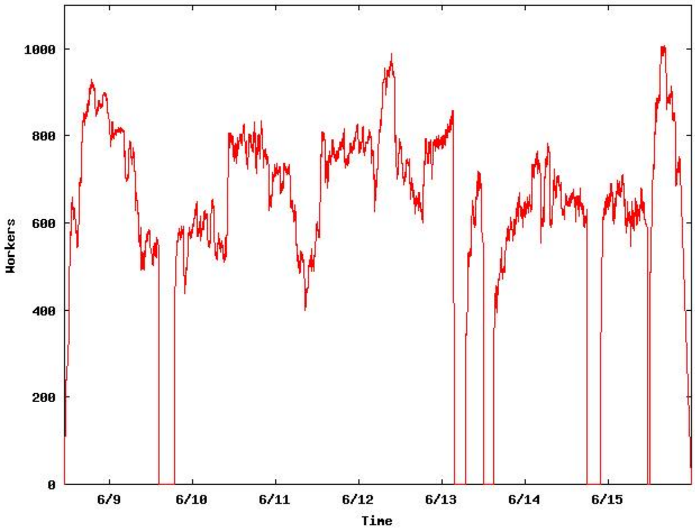
NUG30 is solved!

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

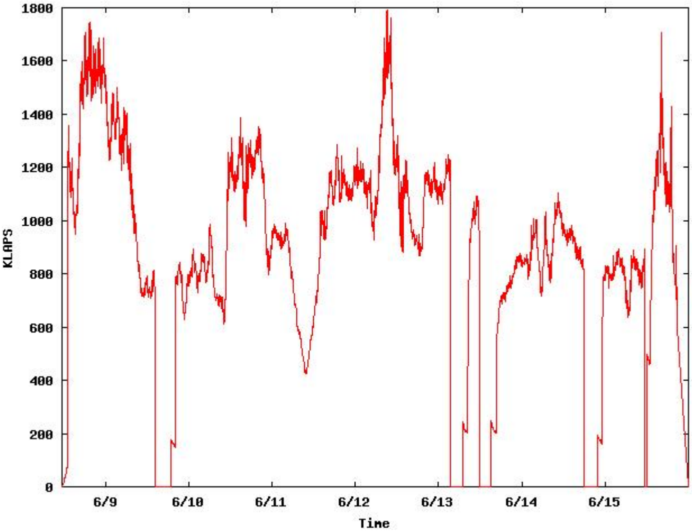
Computation Statistics

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%

MWRollout

- MW (0.1) available from the Condor web page
  - Web: `http://www.cs.wisc.edu/condor/mw`
- Mailing List
  - email `majordomo@cs.wisc.edu` with email body `subscribe mw`
- Bugzilla
  - `http://coral.ie.lehigh.edu/cgi-bin/bugzilla/index.cgi`
  - `mailto:jtl3@lehigh.edu` to become registered user

MWRollout

The Good News!

- MW (0.2). It's getting better and better!
  - Everyone thank Greg Thain!
- Improved robustness: Bug Fixes and Code Scrubbing
- User's Guide
- New (better) examples: knapsack solver with branch-and-bound
- Ported to new platforms: (x86_64, cygwin)
- Part of NMI nightly build procedure

The Bad News!

MW 0.2 will be available "soon"

Conclusions

1. If your parallel algorithm is not "pleasantly" parallel, or requires *dynamic* configuration of tasks, then the master-worker paradigm might be right for you.
2. The master-worker paradigm is very nicely suited to a Grid implementation
   - We really believe that master-worker is the "right" paradigm for distributed computing on the Grid
3. MW can make implementing master-worker algorithms for the Grid easier

Tell Us!

We want YOU to tell us what you want MW to be

1. Easier User Interfaces (C/Python/Java)?
2. Different Communication Interfaces? (MPI?)
3. Support for worker to be "black-box" executable?
4. High-Level Language (matlab/octave), akin to GridSolve?
5. How big do you want to scale?

The End!

We want YOU to join the MW community of users

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

mailto:mw@cs.wisc.edu