An Introduction to the Computational Grid

Jeff Linderoth

Dept. of Industrial and Systems Engineering
Univ. of Wisconsin-Madison
linderot@cs.wisc.edu

COPTA
University of Wisconsin-Madison
October 16, 2007

Outline

- What is “The Grid?”
- Grid Software: Condor, MW
- Large-scale Grid resources: Teragrid, Open Science Grid
- A motivating algorithm: branch-and-bound
- A motivating application: the football pool problem

Come on Let’s Play the Feud

‘‘100 People Surveyed. Top 5 answers are on the board. Here’s the question...’’

Name one common use of the Internet

The Big Board

1. email
2. Looking up answers to homework problems
3. YouTube
4. Updating personal information at myspace
5. Looking at pictures of Anna Kournikova

Strike!

- Doing Computations

- People envision a "Computational Grid" much like the national power grid
- Users can seamlessly draw computational power whenever they need it
- Many resources can be brought together to solve very large problems
- Gives application experts the ability to solve problems of unprecedented scope and complexity, or to study problems which they otherwise would not.
- Large funded initiative in the US.
  - NSF Office of Cyberinfrastructure

Types of Grids

- Computational grids
  - Focus on computationally-intensive operations.
  - This included CPU Scavenging Grids – which is our focus today
- Data grids
  - Help control, share, and manage large quantities of (distributed) data
- Equipment grids
  - Associated with a piece of expensive equipment (telescope, earthquaje shake table, advanced photon source)
  - Grid software used to access and control equipment remotely
- Access grid
  - Used to support group-to-group interactions
  - Consists of multimedia large-format displays, presentation and interactive environments, interfaces to Grid middleware and visualization environments.

Grid Contrasts

(Source: IBM Web Site)

Grid Vs. Web

- Like the web Grid keeps complexity hidden: multiple users enjoy a single, unified experience.
- Unlike the Web which mainly enables communication, grid computing enables full collaboration toward common business or scientific goals.

Grid Vs. P2P

- Like peer-to-peer grid computing allows users to share files.
- Unlike peer-to-peer grid computing allows many-to-many sharing not only files but other resources as well.

Grid Contrasts

Grid Vs. Clusters

- Like clusters and distributed computing, grids bring computing resources together.
- Unlike clusters and distributed computing, which need physical proximity and operating homogeneity, grids can be geographically distributed and heterogeneous.

Grid Vs. Virtualization

- Like virtualization technologies, grid computing enables the virtualization of IT resources.
- Unlike virtualization technologies, which virtualize a single system, grid computing enables the virtualization of vast and disparate IT resources.

This ain't easy!

Read: Nothing works as advertised

- User access and security
  - Who should be allowed to tap in?
- Interfaces
  - How should they tap in?
- Heterogeneity
  - Different hardware, operating systems, and software
- Dynamic
  - Participating Grid resources may come and go
  - Fault-Tolerance is *very* important!
- Communicationally challenged
  - Machines may be very far apart $\Rightarrow$ slow communication.

Grid Computing Tools: Globus

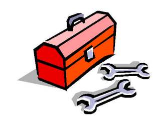
- Globus: Widely-used grid computing toolkit

Globus Services/Libraries

- Security,
- Information infrastructure,
- Resource management,
- Data management,
- Communication,
- Fault detection,
- Portability.

- It is packaged as a set of components that can be used either independently or together to develop applications.

Building a Grid



- Even with wonderful tools like Globus providing these services, there is still a fundamental obstacle to creating computational grids available to all scientists
- GREED!
  - Most people don't want to contribute "their" machine!
- How to induce people to contribute their machine to the Grid?
  - Screensaver – BOINC, `seti@home`
  - Social Welfare – `fightaids@home`
  - Offer frequent flyer miles – company went bankrupt
  - Let the people *keep control* over their machine
  - Give donaters a chance to use the Grid

Condor

THE UNIVERSITY of WISCONSIN MADISON

Peter Couvares
Alan DeSmet
Peter Keller
Miron Livny
Erik Paulsen
Marvin Solomon
Todd Tannenbaum
Greg Thain
Derek Wright

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

Condor: www.cs.wisc.edu/condor

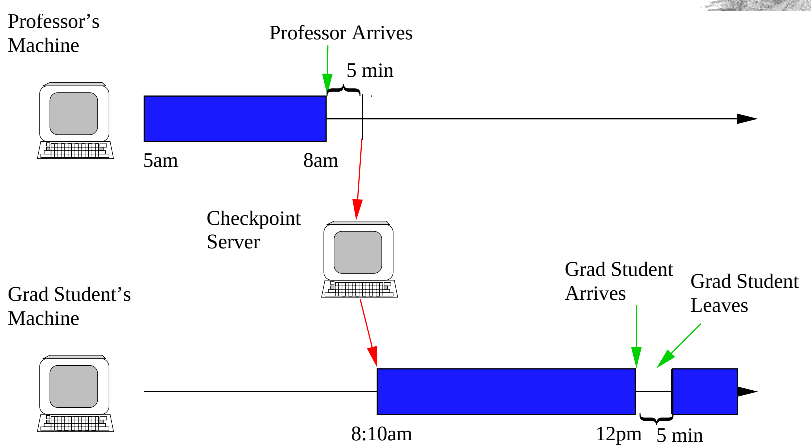
- Manages collections of "distributively owned" workstations
  - User need not have an account or access to the machine
  - Workstation owner specifies conditions under which jobs are allowed to run
  - All jobs are scheduled and "fairly" allocated among the pool
- How does it do this?
  - Scheduling/Matchmaking
  - Jobs can be checkpointed and migrated
  - Remote system calls provide the originating machines environment

Matchmaking

MyType = Job
TargetType = Machine
Owner = ferris
Cmd = cplex
Args = seymour.d10.mps
HasCplex = TRUE
Memory $\geq$ 64
Rank = KFlops
Arch = x86_64
OpSys = LINUX

MyType = Machine
TargetType = Job
Name = nova9
HasCplex = TRUE
Arch = x86_64
OpSys = LINUX
Memory = 256
KFlops = 53997
RebootedDaily = TRUE

Checkpointing/Migration



Other Condor Features

- Pecking Order
  - Users are assigned priorities based on the number of CPU cycles they have recently used.
  - If someone with higher priority wants a machine, your job will be booted off.
- Flocking
  - Condor jobs can negotiate to run in other Condor pools.
- Glide-in
  - Globus provides a "front-end" to many traditional supercomputing sites.
  - Submit a Globus job which creates a temporary Condor pool on the supercomputer, on which users jobs may run.

Condor + Operations Research

- GAMS (`www.gams.com`) has added Grid Computing Language Extensions
- This allows regular GAMS optimization models to be submit to job schedulers like Condor!

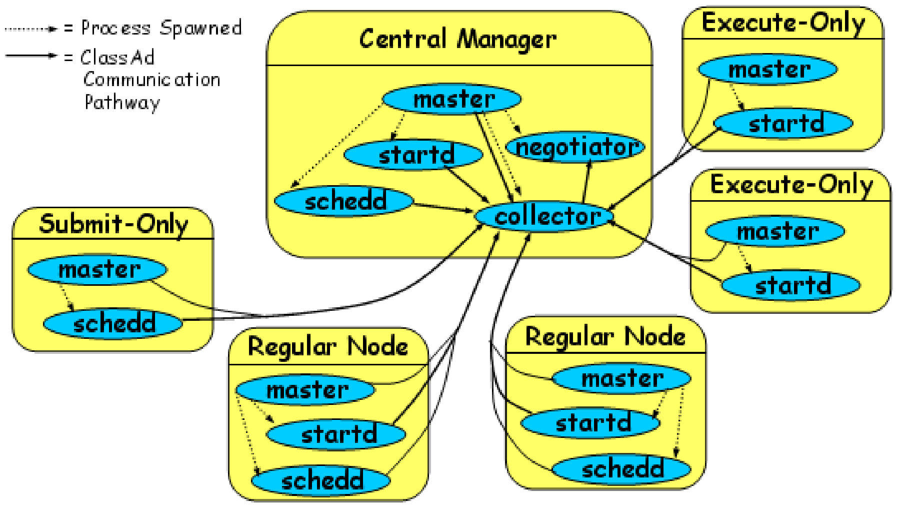
```
mymodel.solvelink=3;
loop(scenario,
  demand=sdemand(scenario); cost=scost(scenario)
  solve mymodel min obj using minlp;
  h(scenario)=mymodel.handle);
```

- Ferris and Busseick use this strategy, in combination with some "manual branching", and CPLEX MIP solver to solve three previously unsolved MIPLIB2003 instances "overnight"
- Stay tuned – next week!

Condor Daemons

- `condor_master`: Controls all daemons
- `condor_startd`: Controls executing jobs
  - `condor_starter`: Helper for starting jobs
- `condor_schedd`: Controls submit jobs
  - `condor_shadow`: Submit-side helper for running jobs
- `condor_collector`: Collects system information; only on Central Manager
- `condor_negotiator`: Assigns jobs to machines; only on Central Manager

A Typical Condor Pool



Building a Grid

Flocking

- Collector from on central manager (`shark.ie.lehigh.edu`) is allowed to negotiate with central manager from a different pool (`condor.cs.wisc.edu`)
- shark's `condor_config`: `FLOCK_TO = condor.cs.wisc.edu`
- condor's `condor_config`: `FLOCK_FROM = shark.ie.lehigh.edu`
- Beware firewalls! (schedd on submit machine must be abe to make direct socket connection to submitting machine)
- There is a tool GCB (Generic Connection Broker) that can get around this limitation

Building a Grid

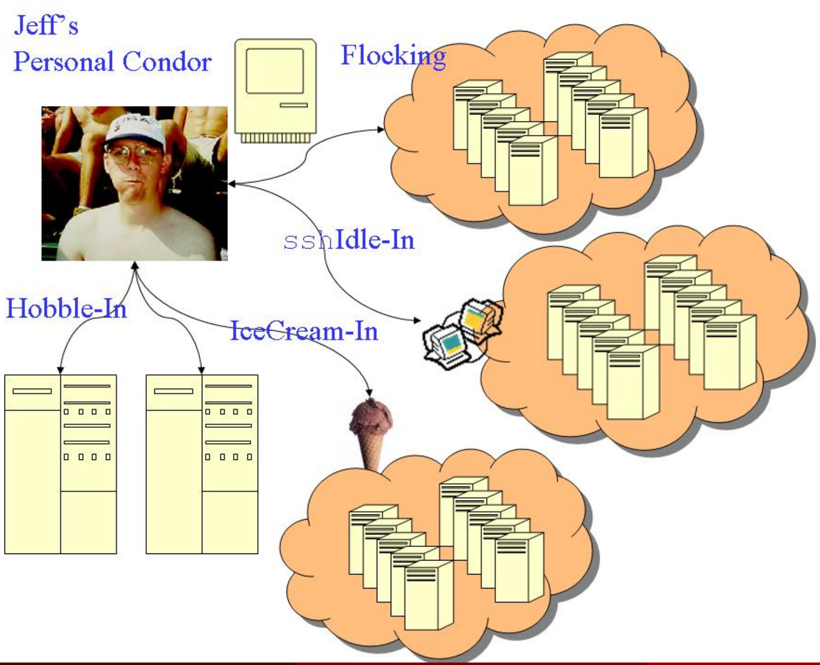
Glide-in

- Often on high-performance computing resource
- Resource request made to gate-keeper
- Gatekeeper make request to batch-scheduled resource.
- When resource is available, startd reports back to central manager, and machine appears as a resource in the "local" condor pool.

Hobble-in

- Forget about trying to use Globus, and do the batch submission of Condor startd's yourself

Personal Condor—A Computational Grid



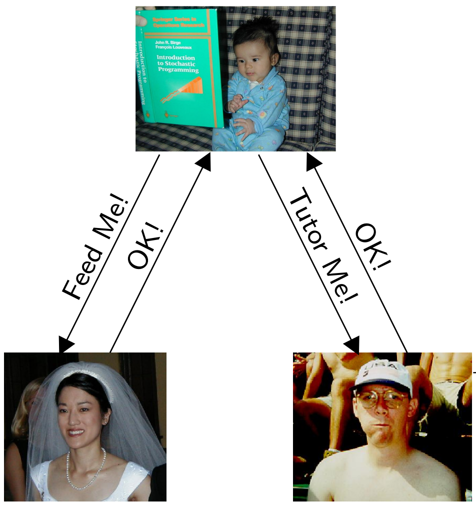
Grid-Enabling Algorithms

- Condor and growing number of interconnection mechanisms gives us the infrastructure from which to build a grid (the spare CPU cycles),
- We still need a mechanism for controlling algorithms 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)
- In response to worker results, the master may generate new tasks (dynamically).

- Simple!
- Fault-tolerant
- Dynamic

Other Important MW Features!

1. Data common to all tasks is sent to workers only once
2. (Try to) Retain workers until the whole computation is complete—don't release them after a single task is done.

These features make for much higher parallel efficiency

- We need to transmit less data between master and workers.
- We avoid the overhead of putting each task on the condor queue and waiting for it to be allocated to a processor.

MW

- Three abstractions in the master-worker paradigm: Master, Worker, and Task.
- The `MW` package encapsulates these abstractions
  - C++ abstract classes
  - User writes 10 functions (Templates and skeletons supplied in distribution)
  - The `MW`ized code will adapt transparently to the dynamic and heterogeneous environment
- The back side of `MW` interfaces to resource management and communications packages:
  - Condor/PVM, Condor/Files
  - Condor/Unix Sockets
  - Single processor (useful for debugging)
  - In principle, could use other platforms.

MW Classes

- MWMaster
  - `get_userinfo()`
  - `setup_initial_tasks()`
  - `pack_worker_init_data()`
  - `act_on_completed_task()`
- MWTask
  - `(un)pack_work`
  - `(un)pack_result`
- MWWorker
  - `unpack_worker_init_data()`
  - `execute_task()`

- Initialization
- Put initial tasks in Master's task list
- Pack(unpack) buffer with data that is sent to worker one time
- Collect results, (maybe) add new tasks
- Pack/unpack work result portions of task
- Does task computation—responsible for filling in results portion for this task

But wait, there's more!

- User-defined checkpointing of master.
  - More compact that Condor checkpoint
  - Must write methods to read/write tasks and master data to file
- (Rudimentary) Task Scheduling
  - MW assigns first task to first idle worker
  - Lists of tasks and workers can be arbitrarily ordered and reordered
  - User can set task rescheduling policies
- User-defined benchmarking
  - A (user-defined) task is sent to each worker upon initialization
  - By accumulating normalized task CPU time, MW computes a performance statistic that is comparable between runs, though the properties of the pool may differ between runs.

MW Applications

- MWKNAP (Glankwamdee, L) – A simple branch-and-bound knapsack solver
- MWFATCOP (Chen, Ferris, L) – A branch and cut code for linear integer programming
- MWQAP (Anstreicher, Brixius, Goux, L) – A branch-and-bound code for solving the quadratic assignment problem
- MWAND (L, Shen) – A nested decomposition-based solver for multistage stochastic linear programming
- MWATR (L, Shapiro, Wright) – A trust-region-enhanced cutting plane code for two-stage linear stochastic programming and statistical verification of solution quality.
- MWSYMCOP (L, Margot, Thain) – An LP-based branch-and-bound solver for symmetric integer programs

The Teragrid http://www.teragrid.org

- Consortium of traditional high-performance computing centers
- $>$ \$150M of NSF funding behind it!
- Over 100 TeraFLOPS! total CPU power
- Dozens of Petabytes of online and archival storage
- 30Gbps backbone

Site	#	Type
IU	712	PowerPC, Itanium, Xeon
NCAR	1024	Blue Gene
SDSC	3612	Itanium, Power-4, Blue Gene
NCSA	4381	Itanium, Altix, Xeon
UC/ANL	316	Itanium, Xeon
CACR	104	Itanium
PSC	5248	Alpha
Purdue	5012	Xeon
TACC	5256	Xeon, Ultra-Sparc
	21,284	

Open Science Grid

- A distributed computing infrastructure for large-scale scientific research, built and operated by a consortium of universities and national laboratories

Computing Resources

- 85 participating institutions
- $\approx$ 25,000 computers.
- 175 TB of storage

"Virtual Organizations"

- Compact Muon Solenoid
- CompBioGrid
- Genome Analysis and Database Update
- Grid Laboratory of Wisconsin
- nanoHUB Network for Computational Nanotechnology

Putting it all together

The Upshot

- You can put all of these components together to solve BIG optimization problems
- You can use byproducts (software tools) of this research
- We still need to use our OR expertise to engineer the algorithms for the computational platform

Branch and Bound for MIP

MIP

$$z_{MIP} \stackrel{\text{def}}{=} \max_{(x,y)\in S} \{c^T x + h^T y\}$$

$$S = \{(x,y) \in \mathbb{Z}_+^{|I|} \times \mathbb{R}_+^{|C|} \mid Ax + Gy \leq b\}$$

$$R(S) = \{(x,y) \in \mathbb{R}_+^{|I|} \times \mathbb{R}_+^{|C|} \mid Ax + Gy \leq b\}$$

Bounds

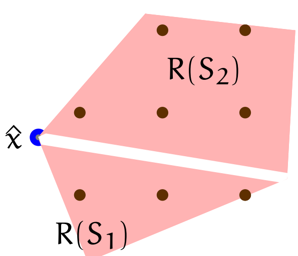
- Upper:

$$z_{LP} \stackrel{\text{def}}{=} \max_{(x,y)\in R(S)} \{c^T x + h^T y\} \geq z_{MIP}$$

- Lower:

$$(\hat{x},\hat{y}) \in S \Rightarrow c^T \hat{x} + h^T \hat{y} \leq z_{MIP}$$

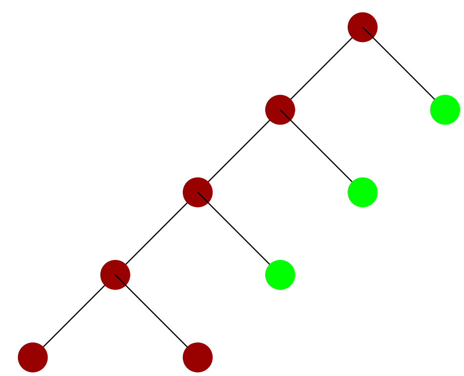
Branch-and-Bound for MIP



1. Solve for z_{LP}, $\hat{x}$
2. Branch: Exclude $\hat{x}$ but no points in S
3. Lather, Rinse, Repeat!

Trees

- Conceptually, this recursive procedure can be arranged into a branch-and-bound tree



Engineering!

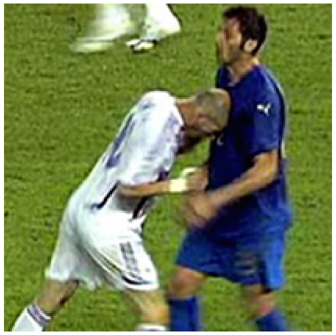
- The way in which you distribute this algorithm on a computational grid can have a huge impact on performance

Performance Tips

- Unit of Work: Subtree (with time cutoff)
- Workers: Search Depth First
- Master:
  - Dynamically adjust grain size depending #workers vs. #tasks
- Master:
  - Dynamically adjust node order, depending on state of memory

Are You Ready for Some Football?!

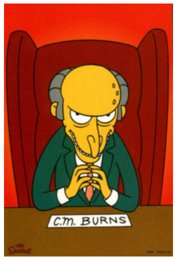
- Predict the outcome of v soccer matches
- $\alpha = 3$
  - 0: Team A wins
  - 1: Team B wins
  - 2: Draw
- You win if you miss at most $d = 1$ games



The Football Pool Problem

What is the minimum number of tickets you must buy to assure yourself a win?

Partners in Crime – Football Pools



{ François Margot
Carnegie Mellon

{ Greg Thain
UW-Madison

How Many Must I Buy?

Known Optimal Values

v	1	2	3	4	5
$\|C^*_v\|$	1	3	5	9	27

The Football Pool Problem

What is $|C^*_6|$?

- Despite significant effort on this problem for > 40 years, it is only known that

$$65 \le C^*_6 \le 73$$

But It's Trivial!

- For each $j \in W$, let $x_j = 1$ iff we word j is in code C
- Let $A \in \{0,1\}^{|W| \times |W|}$ with $a_{ij} = 1$ iff word $i \in W$ is distance $\leq d = 1$ from word $j \in W$

IP Formulation

$$\min e^T x$$

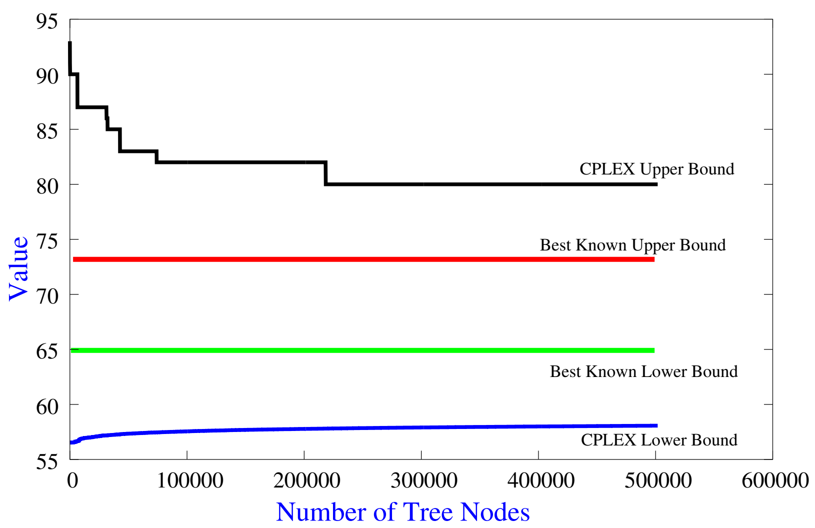
$$\text{s.t.} \quad Ax \geq e$$
$$x \in \{0,1\}^{|W|}$$

CPLEX Can Solve Every IP

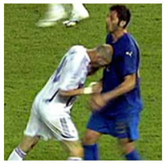
```
         Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt     Gap

      0     0       56.0769   729                     56.0769     2200
*    0+     0                   0      243.0000       56.0769     2200   76.92%
*    0+     0                   0      110.0000       56.0769     2200   49.02%
                    56.5164   729      110.0000    Fract:  56     2542   48.62%
*    0+     0                   0      107.0000       56.5164     2542   47.18%
                    56.5279   729      107.0000     Fract:  6     2673   47.17%
*    0+     0                   0       94.0000       56.5279     2673   39.86%
*    0+     0                   0       93.0000       56.5279     2673   39.22%
Elapsed time =  90.03 sec. (tree size =  0.00 MB)
*   50+    50                   0       91.0000       56.5285    12242   37.88%
Elapsed time = 6841.16 sec. (tree size = 14.12 MB)
  31100 30002       60.1690   544       87.0000       57.1864  5467339   34.27%
  31200 30102       77.7888   216       87.0000       57.1864  5499451   34.27%
* 31200+28950                   0       86.0000       57.1864  5499451   33.50%
  31300 29044       58.9809   611       86.0000       57.1870  5511005   33.50%
Elapsed time = 9500.15 sec. (tree size = 18.70 MB)
  42700 39098       78.3242   197       85.0000       57.2845  7623200   32.61%
* 42740+36552                   0       83.0000       57.2845  7626440   30.98%
Elapsed time = 117349.90 sec. (tree size = 202.88 MB)
Nodefile size = 74.98 MB (61.52 MB after compression)
 465100 434311       66.8425   410       80.0000       58.0439 92473005   27.45%
```


NOT!

- Roughly 10^8 universe lifetimes in order to establish that $|C_6^*| > 72$



Plan of Attack



Apply A Hodgepodge of Tricks

1. **Isomorphism Pruning**: Trick for efficiently ordering search so that nodes that lead to symmetric solutions are not evaluated
2. **Subcode Enumeration**: Enumerate portions of potential codes of cardinality M.
3. **Subcodes and Integer Programming**: Demonstrate (via integer programming) that none of the portions of potential codes leads to a code of size M.
4. **Subcode Sequencing and Variable Aggregation**: The partial solutions can be aggregated and regrouped a bit to lessen the workload
5. **Give it massive computing power**: The Grid!

It Doesn't Sound Like a Good Idea

- After all that hard that hard theoretical and enumerative work, we transformed 1 IP into 1000.

M	# Potential Codes
66	7
67	13
68	45
69	102
70	176
71	264
72	393
	1000

- For a given value of M, solving the related instances establishes that no code C of that cardinality exists
- We solve each of the 1000 IPs on the grid

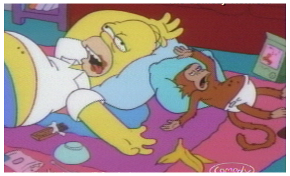
Resources Used in Computation

Site	Access Method	Arch/OS	Machines
Wisconsin - CS	Flocking	x86_32/Linux	975
Wisconsin - CS	Flocking	Windows	126
Wisconsin - CAE	Remote submit	x86_32/Linux	89
Wisconsin - CAE	Remote submit	Windows	936
Lehigh - COR@L Lab	Flocking	x86_32/Linux	57
Lehigh - Campus	Remote Submit	Windows	803
Lehigh - Beowulf	ssh + Remote Submit	x86_32	184
Lehigh - Beowulf	ssh + Remote Submit	x86_64	120
TG - NCSA	Flocking	x86_32/Linux	494
TG - NCSA	Flocking	x86_64/Linux	406
TG - NCSA	Hobble-in	ia64-linux	1732
TG - ANL/UC	Hobble-in	ia-32/Linux	192
TG - ANL/UC	Hobble-in	ia-64/Linux	128
TG - TACC	Hobble-in	x86_64/Linux	5100
TG - SDSC	Hobble-in	ia-64/Linux	524
TG - Purdue	Remote Submit	x86_32/Linux	1099
TG - Purdue	Remote Submit	x86_64/Linux	1529
TG - Purdue	Remote Submit	Windows	1460

OSG Resources Used in Computation

Site	Access Method	Arch/OS	Machines
OSG - Wisconsin	Schedd-on-side	x86_32/Linux	1000
OSG - Nebraska	Schedd-on-side	x86_32/Linux	200
OSG - Caltech	Schedd-on-side	x86_32/Linux	500
OSG - Arkansas	Schedd-on-side	x86_32/Linux	8
OSG - BNL	Schedd-on-side	x86_32/Linux	250
OSG - MIT	Schedd-on-side	x86_32/Linux	200
OSG - Purdue	Schedd-on-side	x86_32/Linux	500
OSG - Florida	Schedd-on-side	x86_32/Linux	100
		OSG:	**2758**
		Total:	19,012

Working Hard!



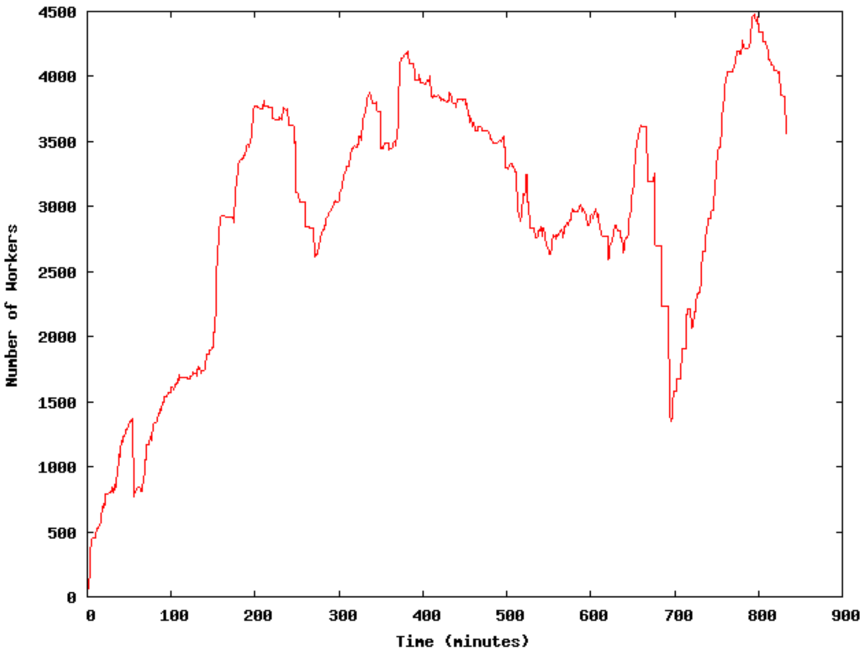
Partial Computational Statistics

	$M = 69$	$M = 70$
Avg. Workers	555.8	562.4
Max Workers	2038	1775
Worker Time (years)	110.1	30.3
Wall Time (days)	72.3	19.7
Worker Util.	90%	82%
Nodes	2.85×10^9	1.89×10^8
LP Pivots	2.65×10^{12}	1.82×10^{11}

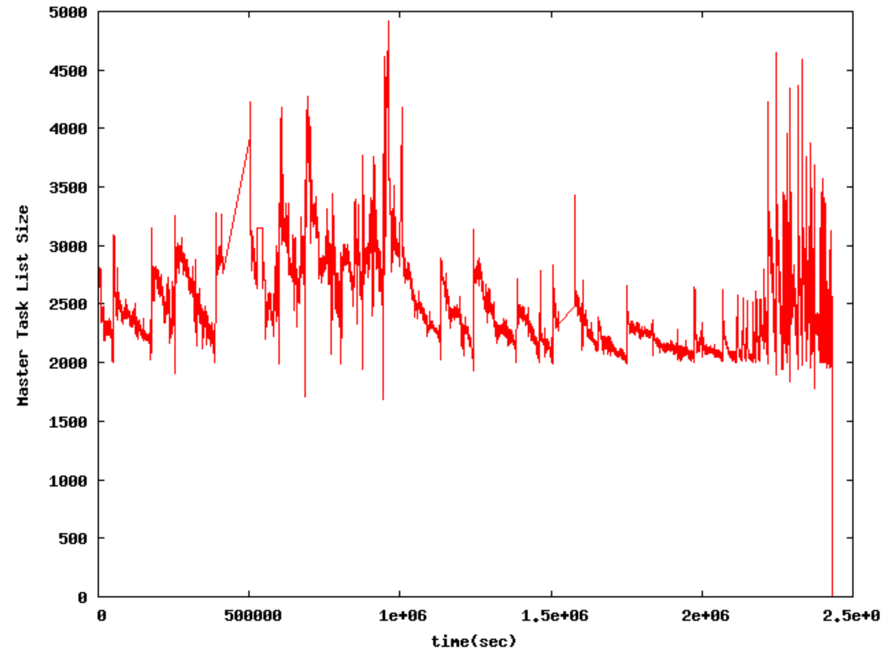
Working on $M = 71$

- Brings the total to > 200 CPU Years!

$M = 71$, Number of Processors (Slice)



$M = 70$, Stack Size (Slice)



Conclusions

The Grid Is Powerful

If you compute in a flexible manner

The Grid is Scalable

If you engineer your algorithm for the platform

We Want You!

To use Condor, MW and "The Grid" for Optimization

- www.cs.wisc.edu/condor
- www.cs.wisc.edu/condor/mw