IE 495 – Lecture 13

Parallel and High Performance Computing for Stochastic Programming

Prof. Jeff Linderoth

February 26, 2003

Outline

- Review
  - Regularlizing
  - Bunching
- Introduction to Parallel Computing
- Using parallel computers to solve stochastic programs

Review

- Why is the LShaped Method Bad?
- What is $\| \cdot \|_\infty$?
- What is the basic idea behind "bunching"?

Bundle-Trust

- *Bundle* — Build up a bundle of subgradients to better approximate your function
- *Trust region* — Stay close (in a region you trust), until you build up a good enough bundle to model your function accurately
  - ◇ Accept new iterate if it improves the objective by a "sufficient" amount. Potentially increase Δ_k. (*Serious Step*)
  - ◇ Otherwise, improve the estimation of $\mathcal{Q}(x^k)$, resolve master problem, and potentially reduce Δ_k (*Null Step*)

⋆ These methods can be shown to converge, *EVEN IF YOU DELETE CUTS.*

Simple Bunching

- Denote by $\mathcal{T} \subseteq \mathcal{R}$ the set of right hand sides that you must solve
- Denote by $\mathcal{U}_k$ the kth "bunch" of right hand sides.

1. Let $\mathcal{T} = \mathcal{R}$. Let $k = 0$
2. Choose a "representative" $r \in \mathcal{T}$. If $\mathcal{T} = \emptyset$, let $b = k$, be the total number of bunches. **Stop.**
3. Solve $\max_\lambda \{\lambda^T r | \lambda^T W \leq q\}$, obtaining a basis B_r, and optimal dual solution λ_k.
4. Forall $t \in \mathcal{T}$, check if $B_r^{-1} t \geq 0$. If so, $\mathcal{U}_k = \mathcal{U}_k \cup t$. Let $\mathcal{T} = \mathcal{T} \setminus \mathcal{U}_k$.
5. **Go To** 2.

High Performance Computing

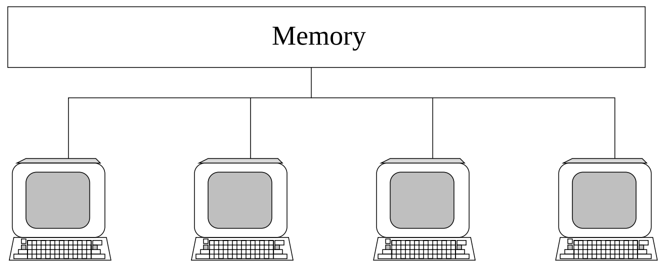
- Bringing together many CPUs to
  - ◇ Solve a problem **faster**
  - ◇ Solve a **bigger** problem

High Performance Computing Uses

- Computational Fluid Dynamics
  - ◇ Climate Modeling
- Finite element and structure analysis
- Numerical solution of (O/P)DE's
- Computational chemistry — chemical kinetics
- Computational biology — protein folding
- Nuclear physics
- Simulations and Monte Carlo Methods
- Optimization!

Types of Parallel Computers — Shared Memory

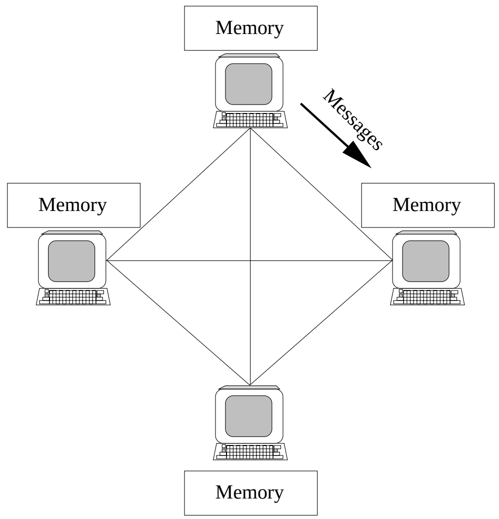
- Shared Memory



- All processors share a common memory (connected by a bus).
- Processes share information by writing and retrieving items from memory.
- Buzzwords: *Multi-Threading*, *openMP*

Types of Parallel Computers — Message Passing

- Message Passing



- Processors connected by a "network"
- They pass messages by sending messages over the network
- Buzzwords: *Sockets, MPI, PVM*

Message Passing Parallel Computers

- Simple "network of workstations" is a message passing parallel computer
  - Buzzword: *COTS*
- Can use more advanced/dedicated hardware to network the computer together
  - Buzzword: *Beowolf Cluster*

A Combination of the Two

- The fastest computers in the world are combinations of these ideas (as well as some other ideas like *vector processors*)
- What is the fastest computer in the world?

`http:www.top500.org`

A New Paradigm—The Grid

- 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 ths ability to solve problems of unprecedented scope and complexity, or to study problems which they otherwise would not.
- *Grid computing* used to be called *Metacomputing*
  - ◇ But now there is a "new" initiative that can be funded!

Separated at Birth?

$\neq$

- There are many ways in which the "Grid" computing I am talking about today is different that the type of parallel (high performance) computing Ted does

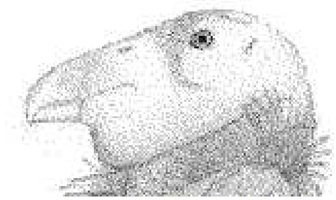
Grid Computing $\neq$ Parallel Computing

- Dynamic
  - ◇ Resources may come and go during the course of the computation
  - $\Rightarrow$ Fault-Tolerance is *very* important!
- Communicationally challenged
  - ◇ Machines may be very far apart
  - $\Rightarrow$ Slow communication channels between them
  - $\Rightarrow$ We prefer CPU-intensive algorithms to data-intensive ones

Grid Computing $\neq$ Parallel Computing

- Larger scale
  - ◇ More resources are potentially available
- Heterogenous
  - ◇ Different hardware, operating systems, and software.
- User access and security
  - ◇ Who (and what) should be allowed to draw from the Grid
- ⋆ Greed!
  - ◇ Most people don't want to contribute "their" machine to the computational pool

What is Condor?



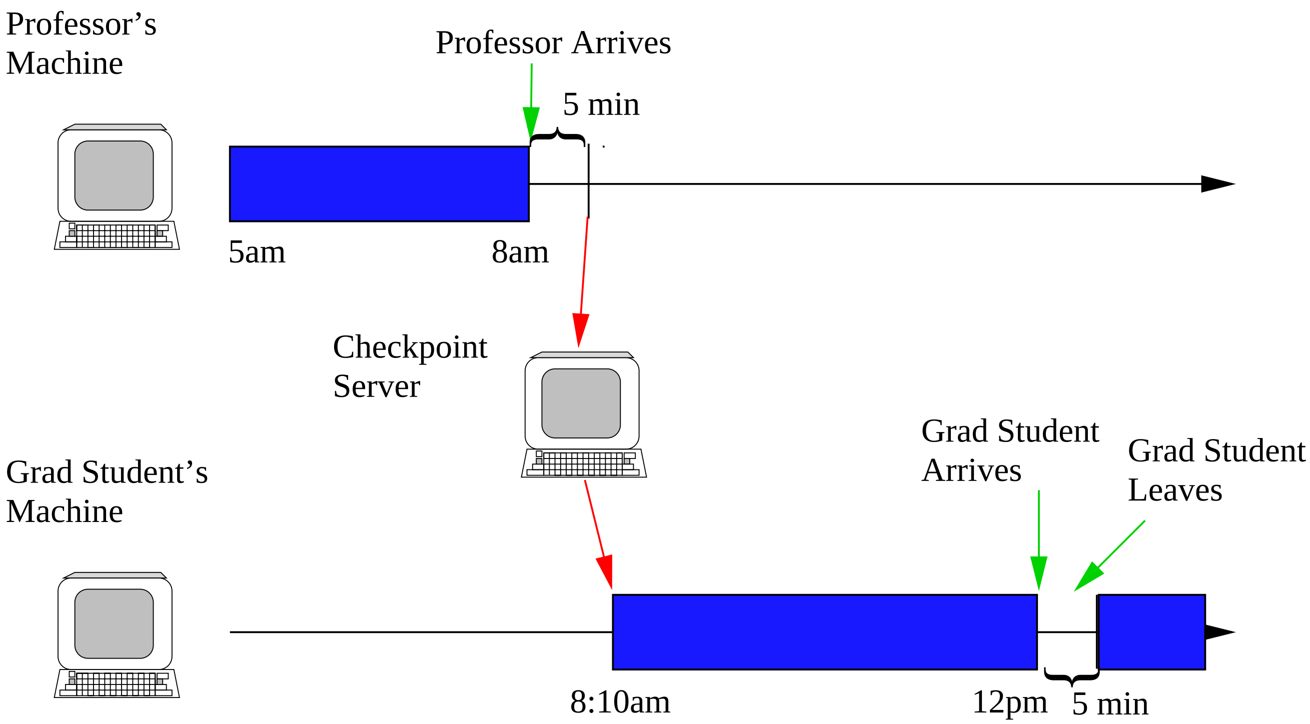
- Manages collections of “distributively owned” workstations
  - User need not have an account or access to the machine
  - Workstation owner specifies conditions under which jobs are allowed to run
  - All jobs are scheduled and “fairly” allocated among the pool
- How does it do this?
  - Scheduling/Matchmaking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

Matchmaking

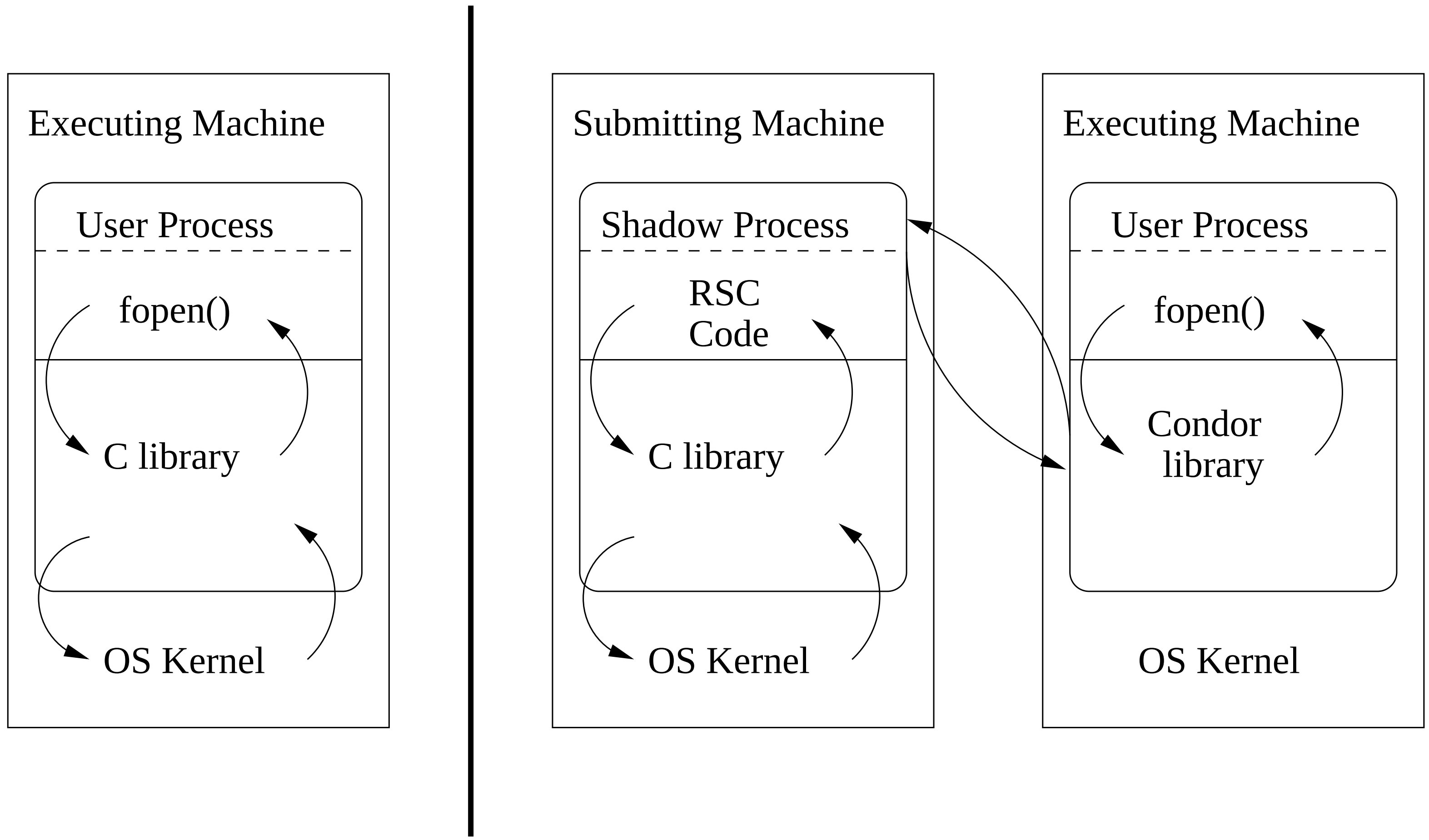
MyType = Job
TargetType = Machine
Owner = ferris
Cmd = cplex
Args = seymour.d10.mps
HasCplex = TRUE
Memory $\geq$ 64
Rank = KFlops
Arch = SUN4u
OpSys = SOLARIS26 | |
SOLARIS27

MyType = Machine
TargetType = Job
Name = nova9
HasCplex = TRUE
Arch = SUN4u
OpSys = SOLARIS27
Memory = 256
KFlops = 53997
RebootedDaily = TRUE

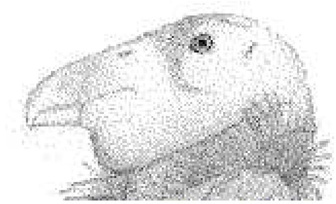
Checkpointing/Migration



Remote System Calls



Other Condor Features



- Pecking Order
  - ◇ Users are assigned priorities based on the number of CPU cycles they have recently used
- Flocking
  - ◇ Condor jobs can negotiate to run in other Condor pools.
- Glide-in
  - ◇ Globus (Grid computing toolkit from Argonne) 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.

Sold!

- I know you all want want to use High-Performance and Grid computing to solve your research problems
- ⋆ There are resources *on campus!*
  - ◇ A small, but ever growing, Condor pool in the ISE Dept
  - ◇ A 32 processor SMP machine (Origin 2000) on Campus
  - ◇ A campus-wide Condor pool (containing the Origin 2000)
  - ⋆ A 128 processor Beowolf Cluster **fire**

- If you think that your research could benefit from more computing resources, *PLEASE LET ME KNOW*.

Parallel Stochastic Optimization

- LOTS of people have done some parallelization of SP
  - ◇ It is an important problem — especially for financial applications
  - ◇ It is relatively easy to parallelize stochastic programming algorithms.
    - + Decomposition-Based
    - + Monte-Carlo (Simulation) Based

References

Ariyawansa, K. A. and D. D. Hudson (1991). Performance of a benchmark parallel implementation of the Van Slyke and Wets algorithm for two-stage stochastic programs on the Sequent/Balance. *Concurrency Practice and Experience 3*, 109–128.

Birge, J. R., C. J. Donohue, D. F. Holmes, and O. G. Svintsitski (1996). A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs. *Mathematical Programming 75*, 327–352.

J. Blomvall and P. O. Lindberg, A Riccati-based primal interior point solver for multistage stochastic programming - extensions, *Optimization Methods and Software*, (2002), pp. 383–407.

Dantzig, G., J. Ho, and G. Infanger (1991, August). Solving stochastic linear programs on a hypercube multicomputer. Technical Report SOL 91-10, Department of Operations Research, Stanford University.

Fragniere, E., J. Gondzio, and J.-P. Vial (2000). Building and solving large-scale stochastic programs on an affordable distributed computing system. *Annals of Operations Research 99*, 167–187.

Gondzio, J. and R. Kouwenberg (1999, May). High performace computing for asset liability management. Technical Report MS-99-004, Department of Mathematics and Statistics, The University of Edinburgh.

Jessup, E., D. Yang, and S. Zenios (1994). Parallel factorization of structured matrices arising in stochastic programming. *SIAM Journal on Optimization 4*, 833–846.

Linderoth, J. T., A. Shapiro, and S. J. Wright (2002, January). The empirical behavior of sampling methods for stochastic programming. Optimization Technical Report 02-01, Computer Sciences Department, University of Wisconsin-Madison.

Linderoth, J. T. and S. J. Wright (2001, April). Decomposition algorithms for stochastic programming on a computational grid. Preprint ANL/MCS-P875-0401, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill. `http://www.optimization-online.org/DB_HTML/2001/04/315.html`.

Mulvey, J. M. and A. Ruszczyński (1995). A new scenario decomposition method for large scale stochastic optimization. *Operations Research 43*, 477–490.

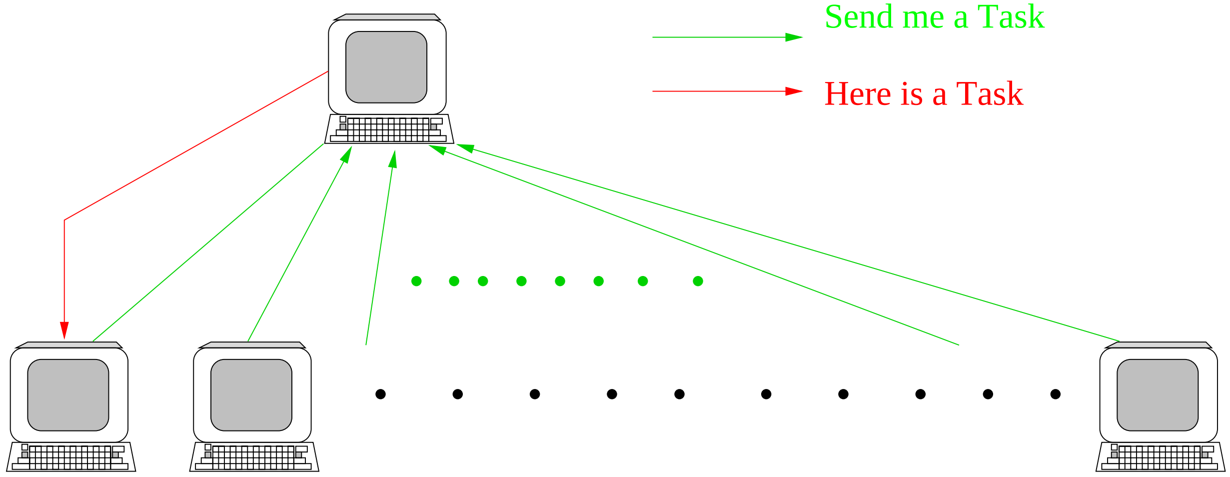
Nielsen, S. S. and S. A. Zenios (1997). Scalable parallel Benders decomposition for stochastic linear programming. *Parallel Computing 23*, 1069–1089.

Ruszczyński, A. (1993). Parallel decomposition of multistage stochastic programming problems. *Mathematical Programming 58*, 201–228.

Parallelizing LShaped

- The evaluation of $\mathcal{Q}(x)$ — solving the different LP's, can be done independently.
  - ◇ If you have K computers, send them each one of K chunks, and your evaluation of $\mathcal{Q}(x)$ will be completed K times faster.
- Work
  - ◇ One or more scenario chunks $\mathcal{S}_{j_1}, \ldots \mathcal{S}_{j_C}$ and point $(\hat{x})$
- Result
  - ◇ A subgradient of each of the $\mathcal{Q}_{[S_k]}(\hat{x})$.
- How many chunks to send in each task?

Contention

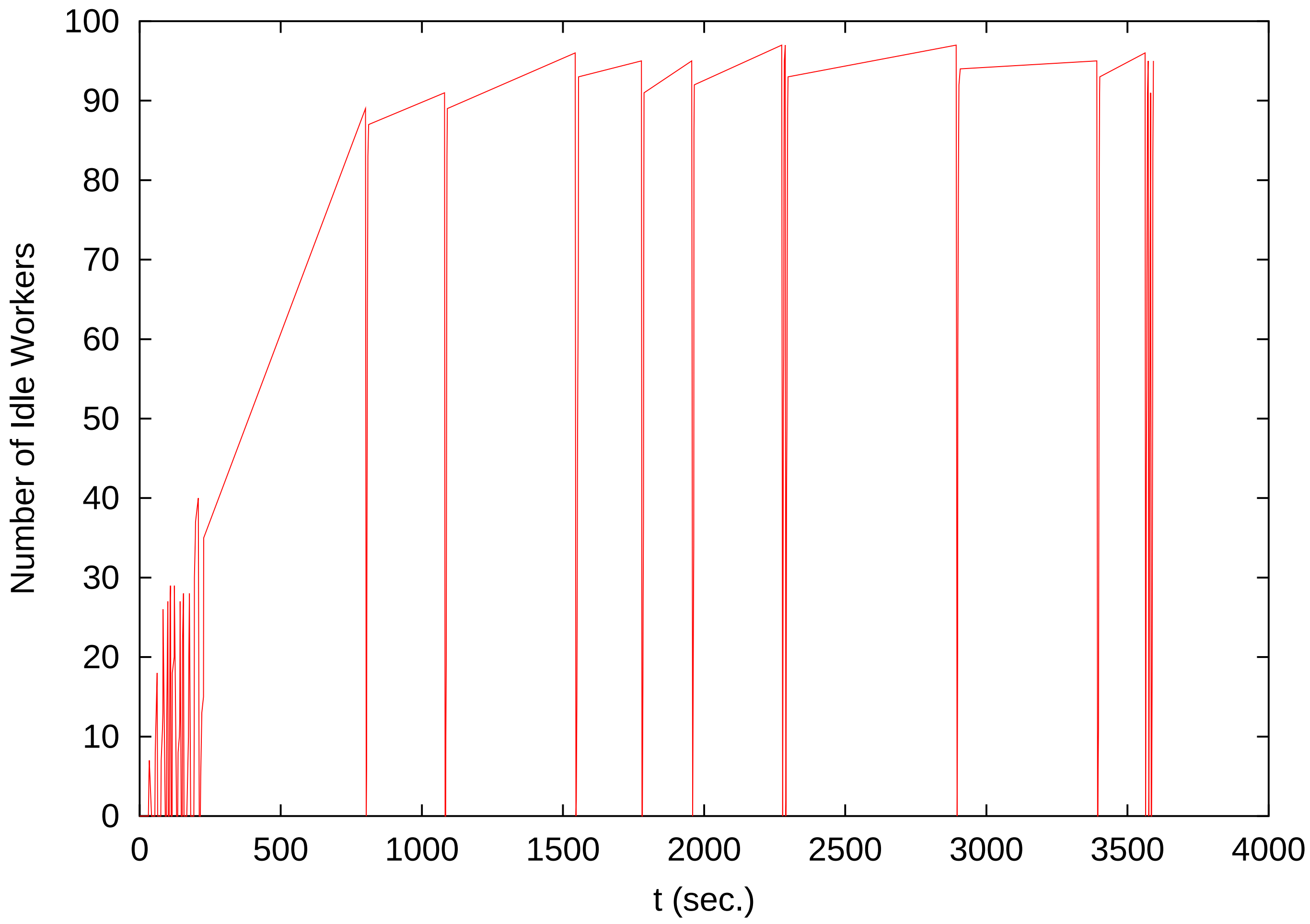


- If task size is too small, the master is overwhelmed with requests, reducing overall efficiency

Other Inefficiencies

- What if solving the master problem takes a long time?
  - ◇ Important to delete cuts
- The "Weakest Link" syndrome

Worker Usage



Stamp Out Synchronicity!

- We start a new iteration only after all "chunks" have been evaluated
  - ◇ On a Grid, different processors act at different speeds, so many may wait idle for the "slowpoke"
  - ◇ Even worse, Grid computing toolkits can fail to inform the user that their worker has failed!
  - ◇ We can never efficiently use more than C^{-1} machines

⋆ Asynchronous methods are preferred for traditional parallel computing environments. They are nearly *required* for Grid computing environments!

ATR – An Asynchronous Trust Region Method

- Keep a "basket" $\mathcal{B}$ of trial points for which we are evaluating the objective function
- Make decision on whether or accept new iterate x^{k+1} after entire $\mathcal{Q}(x^k)$ is computed
- Convergence theory and cut deletion theory is similar to the synchronous algorithm
- Cuts can be deleted if they are no longer relevant to any points in the current basket
- Populate the basket quickly by initially solving the master problem after only α% of the scenario LPs have been solved

ATR

+ *Greatly* reduces the synchronicity requirements
– Might be doing some "unnecessary" work – the candidiate points might be better if you waited for complete information from the preceeding iterations

---

- ALS — Asynchronous LShaped
- TR — Synchronous Trust Region
- ATR — Asynchronous Trust Region

$|S| = 10000$

run	points evaluated	$\|\mathcal{B}\|$	# tasks	cuts/iter (# clusters)	max. processors allowed	av. processors	parallel efficiency	max. # cuts in model	master problem solve time (mi	wall clock time (min
ALS	277	-	10	50	40	32	.31	3089	13.0	453
TR	46	-	10	50	20	19	.35	2301	1.5	119
ATR	117	4	10	50	80	70	.23	3503	3.0	83
ATR	172	9	10	50	80	70	.59	4118	6.5	55
ATR	140	9	20	100	80	68	.80	7048	10.5	43

The World's Largest LP

- Storm – A cargo flight scheduling problem (Mulvey and Ruszczyński)
- We aim to solve an instance with 10,000,000 scenarios
- $x \in \Re^{121}, y(\omega_i) \in \Re^{1259}$
- The deterministic equivalent is of size

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

- # Chunks 1024, $|\mathcal{B}| = 4$
- Started from an $N = 20000$ solution. (*This is what people do in practice, which is why trust region is important*
- $\Delta_0 = 1$

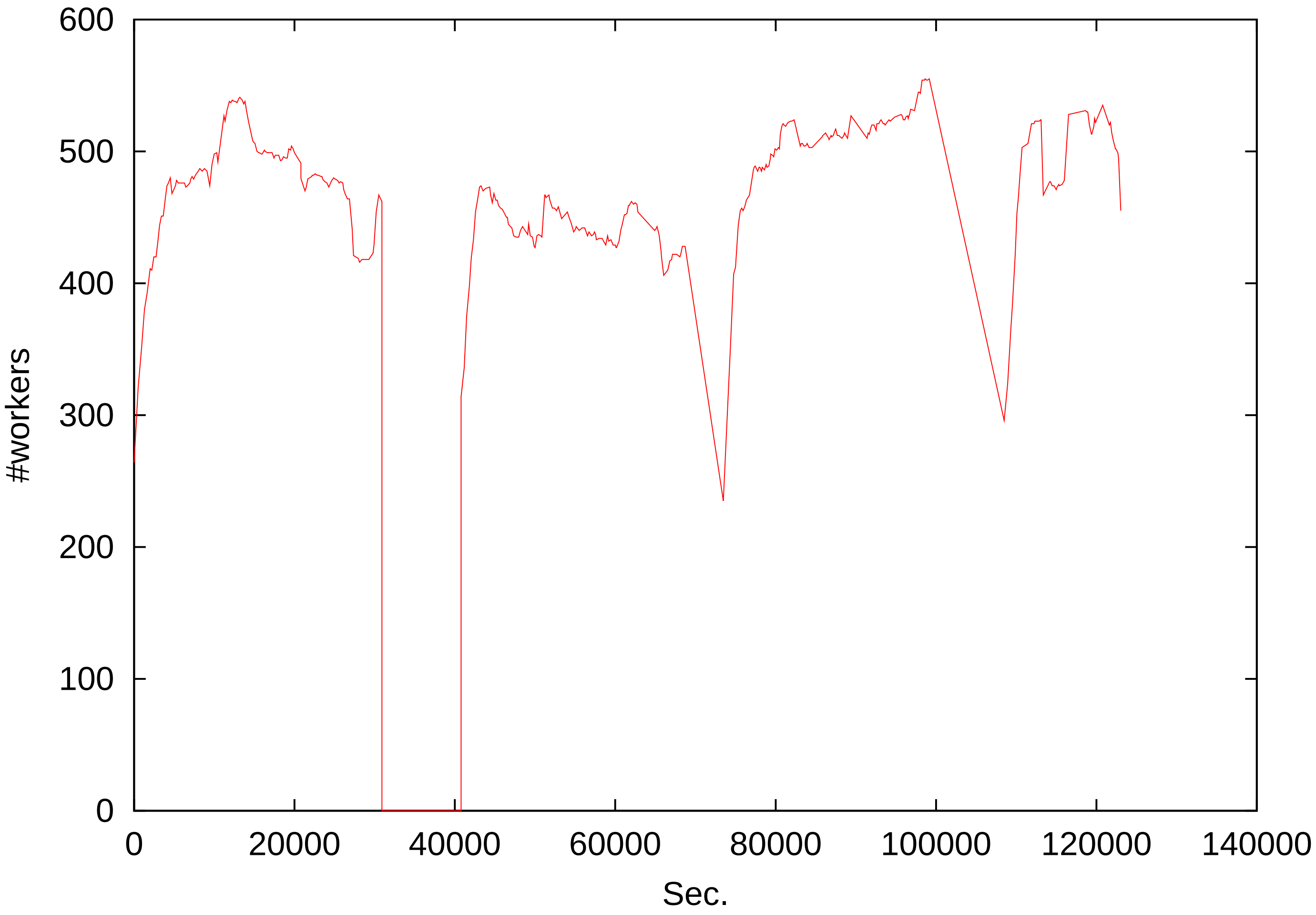
The Super Storm Metacomputer

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



Next Time

- Bounds
- Algorithms Based on Bounds
- HW#2 due
- Will hand out HW#3.
- Project description—due on Wed 3/5.