Condor: Supercomputing Without a Super-Budget

Greg Thain and Steve Wright

University of Wisconsin
Department of Computer Sciences

Institute for Computational Economics, 2006
Outline

1. Introduction to Condor
   - Condor Overview
   - Running your first Condor job
   - Managing Condor jobs

2. Condor Recipes
   - Automatic checkpoint of long-running codes
   - Statistical Bootstrapping
   - DAGMAN: Coordinating dependent jobs
   - Condor and GAMS

3. Master-Worker: Parallel Programming Using Condor
   - Master-Worker
   - An MW Example: Value Function Iteration
   - The World of Condor
Outline

1. Introduction to Condor
   - Condor Overview
   - Running your first Condor job
   - Managing Condor jobs

2. Condor Recipes
   - Automatic checkpoint of long-running codes
   - Statistical Bootstrapping
   - DAGMAN: Coordinating dependent jobs
   - Condor and GAMS

3. Master-Worker: Parallel Programming Using Condor
   - Master-Worker
   - An MW Example: Value Function Iteration
   - The World of Condor
Condor is a cluster computing manager for HTC

HTC: High Throughput Computing

- High Throughput Computing, not High Performance Computing
- Dedicated Clusters
- Cycle scavenging from desktops
Installing Condor

- Call your IT department!
Typical Condor pool

Central Manager
- master
- startd
- negotiator
- schedd
- collector

Execute-Only
- master
- startd

Submit-Only
- master
- schedd

Regular Node
- master
- startd
- schedd

Greg Thain and Steve Wright
Supercomputing with Condor
Three steps to cluster computing

- Prepare your job and inputs
- Write a `submit_file`
- Run and manage your job
Step 1: Prepare your job

- Like going on vacation – pack carefully!
- Check for library and other dependencies
- For checkpointable jobs, run condor_compile
- Gather all inputs together
Step 2: Write a submit file

Submit file describes your jobs to Condor

```
submit_file

universe = vanilla

executable = /usr/bin/matlab
arguments = gonkulate.m

transfer_input_files = gonkulate.m
should_transfer_files = yes
when_to_transfer_output = always

output = out
error = err
log = log
queue 1
```

Greg Thain and Steve Wright
Supercomputing with Condor
Step 3: Submit your job(s)

Shell prompt

# condor_submit submit_file
Submitting job(s).....
Logging submit event(s).....
1 job(s) submitted to cluster 11.
Step 3a: Manage your job(s)

Shell prompt

```bash
condor_rm my_job_number
condor_hold my_job_number
condor_release my_job_number

condor_q
condor_q -run

condor_status
```
Outline

1. Introduction to Condor
   - Condor Overview
   - Running your first Condor job
   - Managing Condor jobs

2. Condor Recipes
   - Automatic checkpoint of long-running codes
   - Statistical Bootstrapping
   - DAGMAN: Coordinating dependent jobs
   - Condor and GAMS

3. Master-Worker: Parallel Programming Using Condor
   - Master-Worker
   - An MW Example: Value Function Iteration
   - The World of Condor
Long running jobs

What if you need to run a job that takes a month to run?
- And the machine crashes?
- Or loses power?
- Or gets rebooted?
Long running jobs

Solution: **Checkpointing!**
- Condor can periodically save the whole state of the job
- And restore it on another machine, if needed
- Some restrictions:
  - Only works on Linux and Solaris (not Windows)
  - You must have source code and be able to relink
  - Several restrictions in functionality
Running Standard Universe Jobs

- `condor_compile` your code
  - `condor_compile` gcc -o solver solver.c
  - `condor_compile` f77 -o executable source.f
- indicate `Standard` universe in your submit file
- Submit as normal
- If execute machine dies, Condor restarts the job elsewhere
- If `submit` machine dies, Condor restarts the job elsewhere
Example: Statistical Bootstrapping
A Parameter Sweep

\{z_1, z_2, z_3, z_4, z_5, ... \} \text{ Distribution} \rightarrow \{z_2, z_2, z_5, ... \} \text{ Sample}

Resamp \{z_2, z_5, z_7, ... \} \rightarrow \text{ Analyze}

Resamp \{z_5, z_7, z_9, ... \} \rightarrow \text{ Analyze}

Resamp \{z_7, z_7, z_9, ... \} \rightarrow \text{ Analyze}

\text{ Coalesce}
Example: Statistical Bootstrapping
A Parameter Sweep

\[ \{z_1, z_2, z_3, z_4, z_5, \ldots\} \quad \text{Distribution} \quad \rightarrow \quad \{z_2, z_2, z_5, \ldots\} \quad \text{Sample} \]

Resamp \{z_2, z_5, z_7, \ldots\}  
Resamp \{z_5, z_7, z_9, \ldots\}  
Resamp \{z_7, z_7, z_9, \ldots\}

Analyze \quad \text{Coalesce} \quad \text{Analyze} \quad \text{Analyze}
Example: Statistical Bootstrapping

A Parameter Sweep

\{Z_1, Z_2, Z_3, Z_4, Z_5, \ldots\} \quad \text{Distribution} \quad \Rightarrow \quad \{Z_2, Z_2, Z_5, \ldots\} \quad \text{Sample}

\text{Resamp} \quad \{Z_2, Z_5, Z_7, \ldots\} \quad \text{Resamp} \quad \{Z_5, Z_7, Z_9, \ldots\} \quad \text{Resamp} \quad \{Z_7, Z_7, Z_9, \ldots\}

\text{Analyze} \quad \text{Analyze} \quad \text{Analyze}

\downarrow

\text{Coalesce}
Example: Statistical Bootstrapping
A Parameter Sweep

\[
\{z_1, z_2, z_3, z_4, z_5, \ldots\} \xrightarrow{\text{Distribution}} \{z_2, z_2, z_5, \ldots\} \xrightarrow{\text{Sample}} \\
\text{Resamp} \{z_2, z_5, z_7, \ldots\} \xrightarrow{} \text{Resamp} \{z_5, z_7, z_9, \ldots\} \xrightarrow{} \text{Resamp} \{z_7, z_7, z_9, \ldots\}
\]

\[
\text{Analyze} \xrightarrow{} \text{Analyze} \xrightarrow{} \text{Analyze} \xrightarrow{\text{Coalesce}}
\]
Example: Statistical Bootstrapping
A Parameter Sweep

\[ \{z_1, z_2, z_3, z_4, z_5, \ldots\} \rightarrow \text{Distribution} \rightarrow \{z_2, z_2, z_5, \ldots\} \rightarrow \text{Sample} \]

\[ \text{Resamp } \{z_2, z_5, z_7, \ldots\} \rightarrow \text{Resamp } \{z_5, z_7, z_9, \ldots\} \rightarrow \text{Resamp } \{z_7, z_7, z_9, \ldots\} \]

\[ \text{Analyze} \rightarrow \text{Coalesce} \rightarrow \text{Analyze} \rightarrow \text{Analyze} \]
Statistical Bootstrapping
A Condor/Matlab implementation

**Driver** Creates distribution.

**Driver** Creates submit file.

**Driver** Runs `condor_submit`.

**Workers** Analyzes subset

**Driver** Processes results.

**driver.m**

```matlab
dist_size = 100000;
d = rand(dist_size, 1) .* 500;
subset = d(floor(rand(1000,1) .* 1000));
save "subset" subset;
```
Statistical Bootstrapping
A Condor/Matlab implementation

Driver Creates distribution.
Driver Creates submit file.
Driver Runs condor_submit.
Workers Analyzes subset
Driver Processes results.

Generated submit_file

```
universe = vanilla
executable = bootclient.m
transfer_files = true
when_to_transfer_output = on_exit
transfer_input_files = subset
output = mean.$(PROCESS)
log = log
queue 5
```
Statistical Bootstrapping
A Condor/Matlab implementation

Driver Creates distribution.
Driver Creates submit file.
Driver Runs condor_submit.
Workers Analyzes subset
Driver Processes results.

```matlab
% driver.m
system("condor_submit file");
system("condor_wait log");
```
Statistical Bootstrapping
A Condor/Matlab implementation

**Driver**
- Creates distribution.
- Creates submit file.
- Runs `condor_submit`.

**Workers**
- Analyzes subset

```matlab
worker.m – All in parallel
load "subset" subset;
subset = subset(floor(rand(10,1) .* 10));
printf("%f ", mean(subset));
```
Statistical Bootstrapping
A Condor/Matlab implementation

**driver.m**

```matlab
while (jobs– > 0)
    tmp = sprintf("mean.%d", jobs);
    f = fopen(tmp, "rb", "native");
    val = fscanf(f, "%f");
    results(jobs + 1) = val;
endwhile
result = mean(results);
```
Running the example

Shell prompt

$ ./bootdriver.m
Submitting job(s).....
Logging submit event(s).....
5 job(s) submitted to cluster 565262.

5 minutes later...

All jobs done.
mean of mean is 161.014978
DAGMan
DAGMan: Directed Acyclic Graph Manager

- Often jobs have dependencies
- One job’s output is another’s input
- Possibly there are many such relationships in your application.

DAGMan solves these problems
Example

- 2-D Matrix of Condor jobs
- Each job has two inputs
  - From leftmost neighbor
  - From lower neighbor
- Initial conditions known
- Desire maximum concurrency

```
41 → 42 → 43 → 44
\uparrow \uparrow \uparrow
31 → 32 → 33 → 34
\uparrow \uparrow \uparrow
21 → 22 → 23 → 24
\uparrow \uparrow \uparrow
11 12 13 14
```
DAGMAN description file

- JOB section names each node and its submit file
- PARENT section describes dependencies
- VARS section names variable to expand in submit file

DAG file

```plaintext
JOB Node_1_1 node.sub
JOB Node_2_1 node.sub
JOB Node_2_2 node.sub

PARENT Node_1_2 Node_2_1 CHILD Node_2_2

VARS Node_2_2 in1="f12"
VARS Node_2_2 in1="f21"
VARS Node_2_2 out="f22"
```
submit

universe = vanilla

executable = sum.pl
arguments = $(IN1) $(IN2) $(IN3)

should_transfer_files = yes
when_to_transfer_output = on_exit

transfer_input_files = $(IN1), $(IN2), $(IN3)

output = $(OUT)
log = log

Notification = never
queue
Running dagman

```bash
$ condor_dag_submit grid.dag
```
Condor and GAMS
Using the Grid support within GAMS

- A new feature of GAMS!
- GAMS itself writes submit files, calls condor_submit
- Uses Condor script to "glue" the piece together
New GAMS Commands

```gams
<model>.solvelink = 3
do not wait for solve, just submit
<model>.handle (set by the 'submitting' solver)
```

```gams
HandleStatus(handle) =
  0  bad handle
  1  model ready to solve but no solution
  2  solution ready to be extracted
```

```gams
executeloadhandle model
; loads all equ and var info
```
Condor and GAMS

Running it

Shell Prompt

```
# gamskeep transgrid10.gms
```
Outlining

1. Introduction to Condor
   - Condor Overview
   - Running your first Condor job
   - Managing Condor jobs

2. Condor Recipes
   - Automatic checkpoint of long-running codes
   - Statistical Bootstrapping
   - DAGMAN: Coordinating dependent jobs
   - Condor and GAMS

3. Master-Worker: Parallel Programming Using Condor
   - Master-Worker
   - An MW Example: Value Function Iteration
   - The World of Condor
Two Condor Shortcomings

- Condor doesn’t run short jobs well.
  - lots of time required to schedule jobs in the pool;
  - time needed to transmit the executable/data/results.
- Condor doesn’t deal directly with parallel algorithms.
  - Can have the process on the user’s workstation generating waves of “worker” jobs to run in parallel, but
    - each worker job must be scheduled anew in the Condor pool, and
    - the master application has to handle all the details.

Master-Worker (MW) addresses these issues!
Master-Worker: Basic Ideas

- 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
- Programming model reusable across many applications.
Other Important Features!

- Data common to all tasks is sent to workers only once.
- (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 now need to transmit much 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.
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 MWized 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**
  - getuserinfo()
  - 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()
But wait there’s more!

- User-defined checkpointing of master. (Don’t lose the whole run if the master crashes.)
- (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.

Greg Thain and Steve Wright
Supercomputing with Condor
MW Applications

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

Given initial capital stock $x_0$, find $V(x_0)$

$$V(x_0) = \left\{ \begin{array}{l}
\max_{(c_t,l_t)} \sum_{t=0}^{\infty} \beta^t u(c_t, l_t) \\
\text{s.t. } x_{t+1} = x_t + f(x_t, l_t) - c_t
\end{array} \right.$$

- $c_t$ and $l_t$ are consumption and labor supply at time $t$
- capital evolves according to $x_{t+1} = x_t + f(x_t, l_t) - c_t$
- $\beta$ is the discount factor and $u(c_t, l_t)$ is the utility given consumption $c_t$ and labor supply $l_t$
- $V(x)$ is the value function for $x_0 = x$
Dynamic Programming

An optimization problem with infinitely many variables: $c_t$, $l_t$, $x_t$, $t = 0, 1, 2, \ldots$, so it’s hard to attack it directly.

But we can use the *dynamic programming principle*, because the optimal objective $V(x_0)$ depends only on $x_0$ - not on any “past history” of $x$.

At the optimal values of $x_t$, $c_t$, $l_t$ we have

\[
V(x_0) = u(c_0, l_0) + \beta \sum_{t=0}^{\infty} \beta^t u(c_{t+1}, l_{t+1}) = u(c_0, l_0) + \beta V(x_1) = u(c_0, l_0) + \beta V(x_0 + f(x_0, l_0) - c_0).
\]

We can use this formula to find $V$ for many different values of $x_0$ simultaneously.
We look for a function $V$ that satisfies this relationship (for all $x$):

$$V(x) = \max_{(c,l)} u(c, l) + \beta V(x + f(x, l) - c).$$

This is the **Bellman equation**.

- The function $V$ is unknown
- Parametric dynamic programming: Approximate $V(x)$ by $\hat{V}(x; a)$, and solve for the parameters $a$ using the Bellman equation.

  - simplest representation: $\hat{V}(x; a) = \sum_{j=0}^{p} a_j x^j$
  - find $a \in \mathbb{R}^{p+1}$ such that $\hat{V}(x; a)$ “approximately” satisfies the Bellman equation, on a finite grid of $x$ values: $x^1, x^2, \ldots, x^n$. (Data Fitting.)
Value Function Iteration

Step 0. *Initialization.* Choose functional form for $\hat{V}(x; a)$ and approximation grid $X = \{x_1, \ldots, x_n\}$. Make initial guess $\hat{V}(x; a^0)$ and choose $\epsilon > 0$.

Step 1. *Maximization step.* Fix $a^k = (a^k_j)_{j=1}^p$. For $i = 1, \ldots, n$, compute

$$v_i = T\hat{V}^k(x_i, a^k) = \max_{(c_i, l_i)} u(c_i, l_i) + \beta \hat{V}(x_i + f(x_i, l_i) - c_i, a^k)$$

Step 2. *Data Fitting for $a$:* Fix $c, l$. Find $a^{k+1}$ s.t.

$$a^{k+1} = \arg \min_a \| \hat{V}(x, a) - v \|^2$$

Step 3. *Convergence.* If $\| \hat{V}(x, a^{k+1}) - \hat{V}(x, a^k) \|_\infty > \epsilon$, set $k \leftarrow k + 1$ and go to Step 1; otherwise stop and report solution.
Value Function Iteration in MW

**MASTER:** *Initialization.* Choose functional form for \( \hat{V}(x; a) \) and approximation grid \( X = \{x_1, \ldots, x_n\} \).

Make initial guess \( \hat{V}(x; a^0) \) and choose \( \epsilon > 0 \).

**WORKER:** *Maximization:* Fix \( a^k = (a^k_j)_{j=1}^p \).

For \( i = 1, \ldots, n \), compute (in parallel)

\[
v_i = T \hat{V}^k(x_i, a^k) = \max_{(c_i, l_i)} u(c_i, l_i) + \beta \hat{V}(x_i + f(x_i, l_i) - c_i, a^k)
\]

**MASTER:** *Data Fitting:* Fix \( c, l \). Find \( a^{k+1} \) s.t.

\[
a^{k+1} = \arg \min_a \| \hat{V}(x, a) - v \|^2
\]

**MASTER:** *Convergence.* If \( \| \hat{V}(x, a^{k+1}) - \hat{V}(x, a^k) \|_{\infty} > \epsilon \), set \( k \leftarrow k + 1 \) and go to Step 1; otherwise stop.
MW Implementation

Each task finds the optimal \((c_i, l_i)\) for a batch of \(x_i\)’s.

- Calls a simple FORTRAN code (to demonstrate that we can!) to do minimizations.
- Hot starting: the optimal \((c_i, l_i)\) is usually a great starting point for \((c_{i+1}, l_{i+1})\)—so report these values to the master for use at the next iteration.
- The task wrapper (C++) and the FORTRAN code communicate via files.

Good algorithms are still vitally important!
A smart, hard-to-parallelize algorithm often beats a dumb, pleasantly parallel algorithm.
act_on_complete_task() on the Master stores the $v_i$’s (and the $c_i$ and $l_i$ values) as they arrive from the workers. When all workers have reported, it solves the least-squares problem (fitting step) to find $a^{k+1}$.

- Could still take a fitting step without waiting for all tasks to report (partial information) to avoid hangups if some workers go down.
- Could adapt size of task (number of $x_i$’s in each task) to accommodate workers of different speeds.
Judd: These models can get very big!!!

- **Investment Portfolio**
  - $d$ assets in the portfolio
  - $X_j = \{x_{j1}, \ldots, x_{jn}\}$ represents $j$-th asset’s position
  - state space: $X = X_1 \times X_2 \cdots \times X_d$
  - transaction cost occurs when adjusting asset positions

- **Dynamic Principal-Agent Problem**
  - the CEO’s performance is evaluated by multiple measures, e.g. stock price, annual profits, etc.
  - the company decides the CEO’s compensation package

- Many other economic applications
MWGAMS
Running short-lived GAMS as a MW task

- MWGAMS is a MW application which runs GAMS in the worker
- Good for jobs with a lot of short optimization problems
- User writes entirely in GAMS – no C++ code at all
Summary

- Condor can easily manage dedicated and desktop machine
- Idle workstations can provide lots of compute cycles
- Master - Worker is a good way to run massively parallel applications
Condor team
Condor in the US

Condor World Map
http://www.cs.wisc.edu/condor/map
Mon Jun 19 18:45:28 CDT 2006

Greg Thain and Steve Wright
Supercomputing with Condor
Condor in Europe

Condor World Map
http://www.cs.wisc.edu/condor/map
Mon Jun 19 18:45:18 CDT 2006

The World of Condor

Summary

The World of Condor

Condor Recipes

Mon Jun 19 18:45:18 CDT 2006

Master-Worker: Parallel Programming Using Condor

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

Greg Thain and Steve Wright

Supercomputing with Condor
Condor in the World

Condor World Map
http://www.cs.wisc.edu/condor/map
Mon Jun 19 18:44:52 CDT 2006

Greg Thain and Steve Wright
Supercomputing with Condor
For more information

- Talk to Greg or Steve!
  - gthain@cs.wisc.edu
- Condor web site: http://www.cs.wisc.edu
- Condor-users mailing list (see web site)
- Condor Week 2007
- (If all else fails) 600 page condor manual
- Talk to Miron re: collaboration