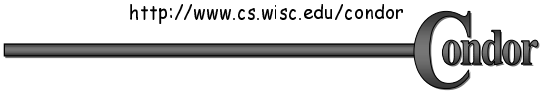
Remote I/O in Condor

Douglas Thain
Computer Sciences Department
University of Wisconsin-Madison
thain@cs.wisc.edu
http://www.cs.wisc.edu/condor



Outline

- **Introduction**
- Using Remote I/O
- Under the Hood
- Build Your Own: Bypass
- Conclusion

Introduction

- The National Technology Grid provides you with access to a diverse array of machines.
- Although you have permission to use these machines, they may be unfriendly to your application.

Introduction (Cont.)

- Remote I/O is an adapter which provides a friendly execution environment on an unfriendly machine.
- Condor uses remote I/O to homogenize the many machines in a Condor pool.
- Can we adapt this to the Grid?

What is Unfriendly?

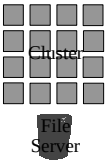
- Programs can technically execute:
  - Correct CPU and OS and enough memory
- But missing some critical items:
  - No input files.
  - No space for output files.
  - No shared filesystem.
  - No login - run as "nobody"?

Range of Unfriendliness

- Anonymous compute node on the Grid:
  - Run as "nobody", with no access to disk.
- Machine at other institution:
  - Can login, have some disk, but no file system.
- Machine down the hall:
  - Can login, share one NFS mount, but not another.

Why use an unfriendly machine?

- After all, homogeneous clusters are the norm:
  - 10s or 100s of identical machines.
  - Centrally administrated.
  - Shared filesystem

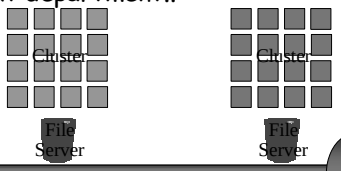


Condor

www.cs.wisc.edu/condor

Need more machines!

- Another hundred idle machines could be found across the street or in the next department..

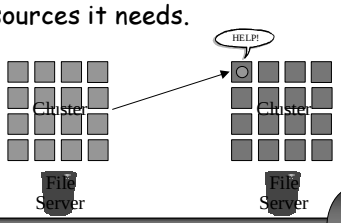


Condor

www.cs.wisc.edu/condor

Need more machines! (Cont.)

- But, your application may not find the resources it needs.

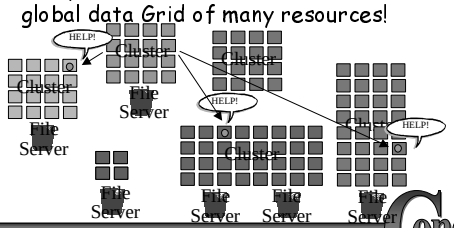


Condor

www.cs.wisc.edu/condor

Need more machines! (Cont.)

- The problem is worse when we consider a global data Grid of many resources!

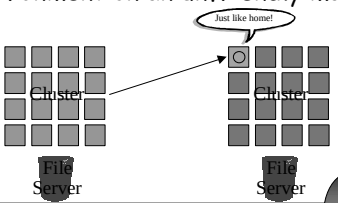


Condor

www.cs.wisc.edu/condor

Solution: Remote I/O

- Condor remote I/O creates a friendly environment on an unfriendly machine.



Condor

www.cs.wisc.edu/condor

Outline

- Introduction
- **Using Remote I/O**
- Under the Hood
- Build Your Own: Bypass
- Conclusion

Condor

www.cs.wisc.edu/condor

Using Remote I/O

- Condor provides several "universes":
  - Vanilla - UNIX jobs + do not need remote I/O
  - Standard - UNIX jobs + remote I/O
  - Scheduler - UNIX job on home machine
  - Globus - UNIX jobs -> Globus
  - PVM - specialized PVM jobs

Vanilla Universe

- Submit any sort of UNIX program to the Condor system.
- Pros:
  - No relinking required.
  - Any program at all, including
    - Binaries
    - Shell scripts
    - Interpreted programs (java, perl)
    - Multiple processes

Vanilla Universe (Cont.)

- Cons:
  - No checkpointing.
  - Very limited remote I/O services.
    - Specify input files explicitly.
    - Specify output files explicitly.
  - Condor will refuse to start a vanilla job on a machine that is unfriendly.
    - ClassAds: FilesystemDomain and UIDDomain

Standard Universe

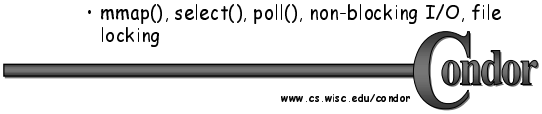
- Submit a specially-linked UNIX application to the Condor system.
- Pros:
  - Checkpointing
  - Remote I/O services:
    - Friendly environment anywhere in the world.
    - Data buffering and staging.
    - I/O performance feedback.
    - User remapping of data sources.

Standard Universe (Cont.)

- Cons:
  - Must statically link with Condor library.
  - Limited class of applications:
    - Single-process UNIX binaries.
    - A number of system calls prohibited.

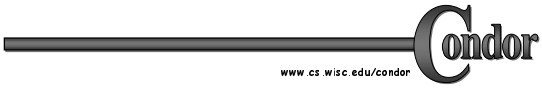
System Call Limitations

- Standard universe does not allow:
  - Multiple processes:
    - fork(), exec(), system()
  - Inter-process communication:
    - semaphores, messages, shared memory
  - Complex I/O:
    - mmap(), select(), poll(), non-blocking I/O, file locking



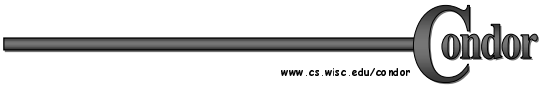
System Call Limitations (Cont.)

- Standard universe also does not allow:
  - Kernel-level threads.
- Too restrictive? Try the vanilla universe.



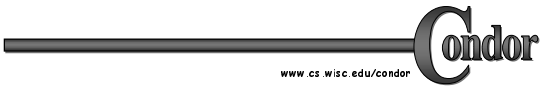
System Call Features

- The standard universe does allow:
  - Signals
    - But, Condor reserves SIGTSTP and SIGUSR1.
  - Sockets
    - Keep it brief – network connections, by nature, cannot migrate or checkpoint.



System Call Features (Cont.)

- The standard universe does allow:
  - Complex I/O on sockets
    - select(), poll(), and non-blocking I/O can be used on sockets, but not other sorts of files.
  - User-level threads



What Universe?

- Vanilla:
  - Perfect for a Condor pool of identical machines.
- Standard:
  - Needed for heterogeneous Condor pools, flocked pools, and more generally, unfriendly machines on the Grid.
- The rest of this talk concerns the standard universe.

Using the Standard Universe

- Link with Condor library.
- Submit the job.
- Get brief I/O feedback while running.
- Get complete I/O feedback when done.
- If needed, remap files.

Link with Condor Library

- Simply use condor_compile in front of your normal link line.
- For example,

```
gcc main.o utils.o -o program
```

- Becomes:

```
condor_compile gcc main.o utils.o -o program
```

- Despite the name, only re-linking is required, not re-compiling.

Submit Job

- Create a submit file:

```
% vi program.submit
```

```
Universe = standard

input = program.in
output = program.out

executable = program

queue 3
```

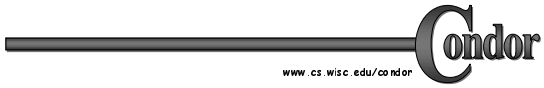
- Submit the job:

```
% condor_submit program.submit
```

www.cs.wisc.edu/condor

Brief I/O Summary

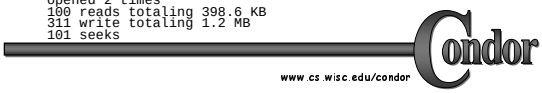
```
% condor_q -io

-- Schedd: c01.cs.wisc.edu : <128.105.146.101:2016>
ID      OWNER         READ    WRITE    SEEK    XPUT     BUFSIZE  BLKSIZE
756.15  joe       244.9 KB 379.8 KB      71   1.3 KB/s 512.0 KB  32.0 KB
758.24  joe       198.8 KB 219.5 KB      78  45.0 B /s 512.0 KB  32.0 KB
758.26  joe        44.7 KB  22.1 KB    2727  13.0 B /s 512.0 KB  32.0 KB

3 jobs; 0 idle, 3 running, 0 held
```



www.cs.wisc.edu/condor

Complete I/O Summary in Email

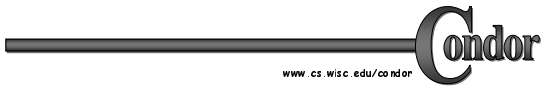
```
Your condor job "/usr/joe/records.remote input output" exited
with status 0.

Total I/O:
      104.2 KB/s effective throughput
      5 files opened
      104 reads totaling 411.0 KB
      316 writes totaling 1.2 MB
      102 seeks

I/O by File:

buffered file /usr/joe/output
      opened 2 times
      4 reads totaling 12.4 KB
      4 writes totaling 12.4 KB

buffered file /usr/joe/input
      opened 2 times
      100 reads totaling 398.6 KB
      311 write totaling 1.2 MB
      101 seeks
```



www.cs.wisc.edu/condor

Complete I/O Summary in Email

- The summary helps identify performance problems. Even advanced users don't know *exactly* how their programs and libraries operate.



www.cs.wisc.edu/condor

Complete I/O Summary in Email (Cont.)

- Example:
  - CMS - physics analysis program.
  - "Why is this job so slow?"
  - Data summary: read 250 MB from 20 MB file.
  - Very high SEEK total -> random access.
  - Solution: Increase data buffer to 20 MB.

www.cs.wisc.edu/condor

Buffer Parameters

- By default:
  - buffer_size = 524288 (512 KB)
  - buffer_block_size = 32768 (32 KB)
- Change parameters in submit file:
  - buffer_size = 20000000
  - buffer_block_size = 32768

www.cs.wisc.edu/condor

If Needed, Remap Files

- Suppose the program is hard-coded to open `datafile`, but you want each instance to get a slightly different copy. In the submit file, add:

```
file_remaps = "datafile = /usr/joe.data.$(PROCESS)"
```

- Process one gets

```
/usr/joe.data.1
```

- Process two gets

```
/usr/joe.data.2
```

- And so on…

If Needed, Remap Files (Cont.)

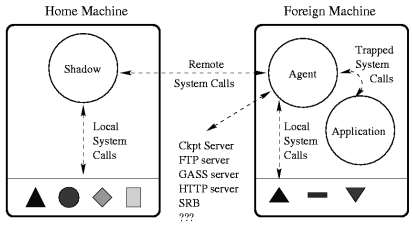
- The same syntax will allows the user to direct the application to other third-party data sources such as web servers:

```
file_remaps = "datafile =
   http://www.cs.wisc.edu/usr/joe/data"
```

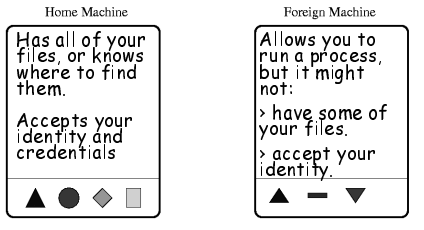

Outline

- Introduction
- Using Remote I/O
- **Under the Hood**
- Build Your Own: Bypass
- Conclusion

The Big Picture



The Machines

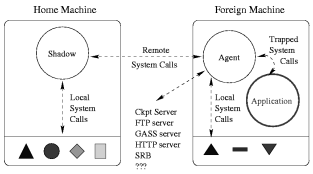


General Strategy

- Trap all the application's I/O operations.
  - open(), close(), read(), write(), seek(), …
- Route them to the right service (at the shadow's direction)
- Cache both service decisions and actual data.

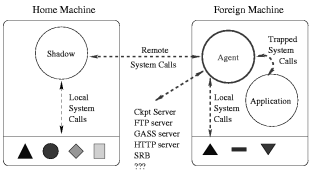
Application

- Plain UNIX program.
- Unaware that it is part of a distributed system.
- Statically linked against Condor library.



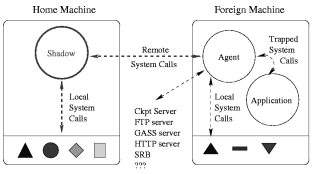
Condor Library

- Sends system calls to various services via RPC.
- Buffers and stages data.
- Asks shadow for policy decisions.

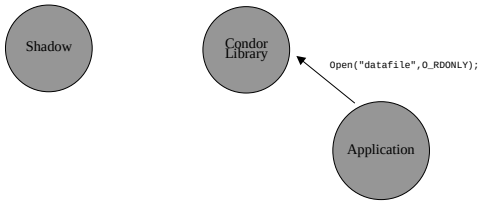


Shadow

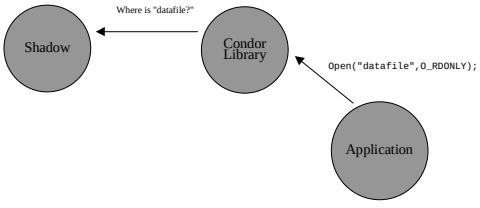
- Makes policy decisions for application.
- Executes remote system calls for application.



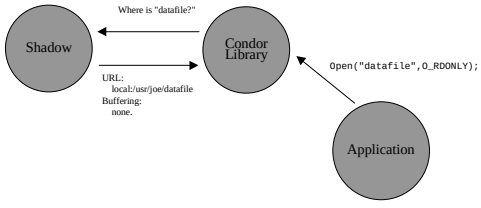
Opening a File



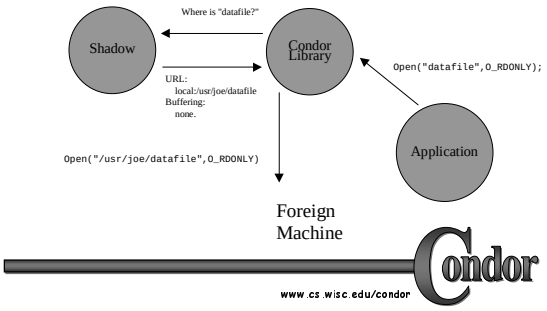
Opening a File



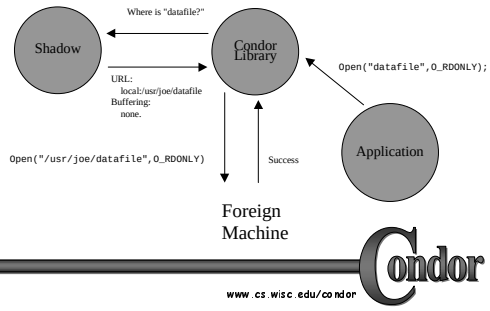
Opening a File



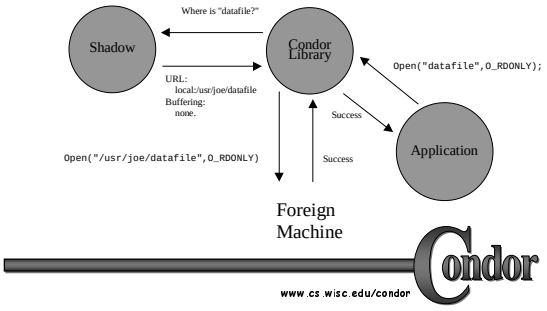
Opening a File



Opening a File



Opening a File



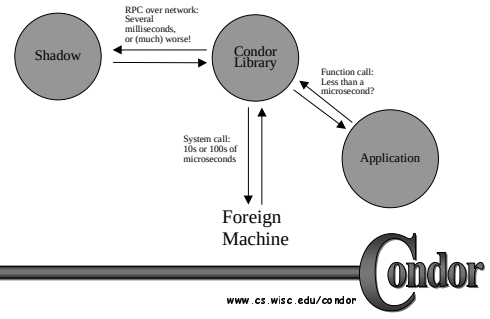
Shadow Responses

- URL:
  - remote: Use remote system calls.
  - local: Use local system calls.
  - special: Use local system calls, disable checkpointing.
  - http: Fetch from a web server.
  - Others in development…

Shadow Responses (Cont.)

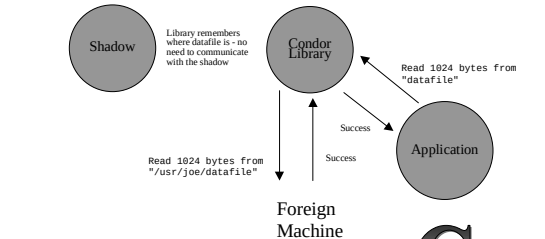
- Buffering:
  - None.
  - Buffer partial data.
  - Stage whole file to local disk.

Reading data from a file



Reading data from a file

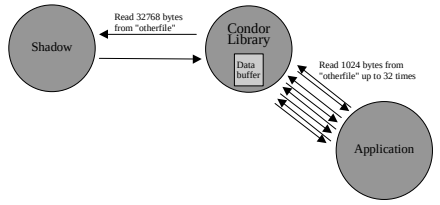
Low latency, random-access data source: Read directly



www.cs.wisc.edu/condor

Reading data from a file

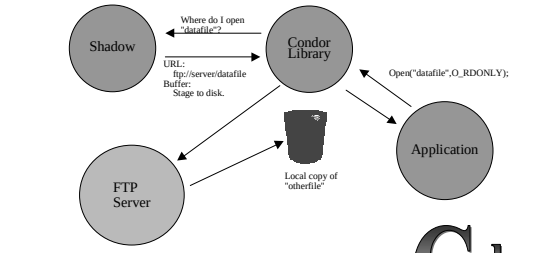
High-latency, random-access data source: Buffer large chunks



www.cs.wisc.edu/condor

Reading data from a file

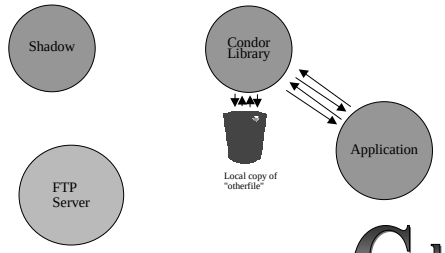
High-latency, sequential-access data source: Stage file to local disk.



www.cs.wisc.edu/condor

Reading data from a file

Random access service can be provided from the local copy.



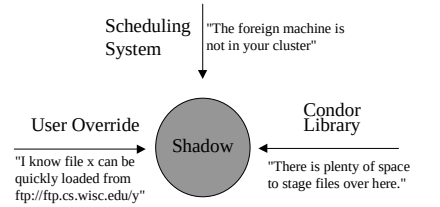
www.cs.wisc.edu/condor

Guiding Principle

- Policy in shadow, mechanisms in library.
  - Shadow makes policy decisions because it knows the system configuration.
  - Library is closest to the application, so it routes system calls to the destination selected by the shadow.

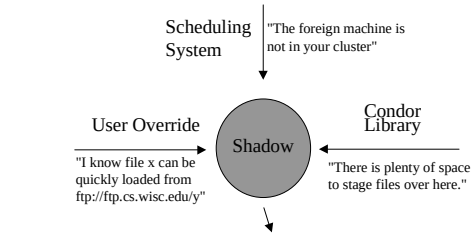
www.cs.wisc.edu/condor

Policy at Shadow



www.cs.wisc.edu/condor

Policy at Shadow



Policy Decisions

- May be different on each foreign machine
  - In same building: "use foreign machine"
  - In other country: "use home machine"
- May change as job migrates
  - same building -> other country
- May change by user control
  - "Let's see if NFS is faster than AFS"

Outline

- Introduction
- Using Remote I/O
- Under the Hood
- **Build Your Own: Bypass**
- Conclusion

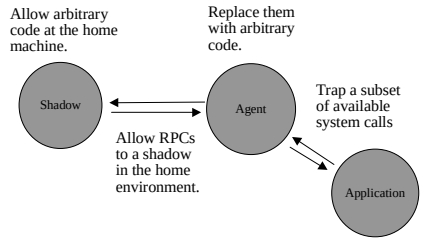
Build Your Own: Bypass

- Generalize remote I/O -> split execution.
- Building split execution systems is hard.
- Bypass is a tool for building split execution systems.

Build Your Own: Bypass (Cont.)

- Unlike Condor, Bypass can be used on any UNIX program without re-linking.
- Example: GASS Agent

Generalized Split Execution



Split Execution is Hard

- Trapping system calls involves a large body of knowledge of particular OS and version
  - Library entry points:
    - _read, __read, __libc_read
  - System call entries:
    - socket(), open("/dev/tcp")
  - Wacky header files:
    - #define stat(a,b) _xstat(VERSION,a,b)

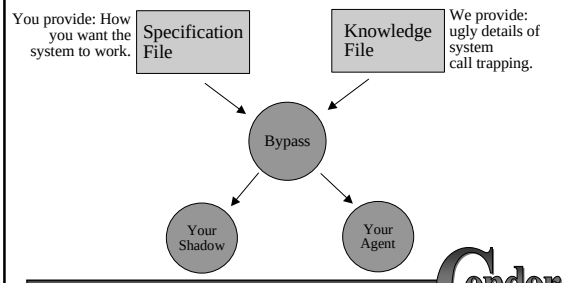
Split Execution is Hard (Cont.)

- RPCs must be platform-neutral
  - Byte sizes and ordering
    - off_t is 8 bytes on Alpha, but 4 bytes on Intel
  - Structure contents and order
    - struct stat has different members on different platforms
  - Symbolic values
    - O_CREAT is a source-level symbol, but its actual value is different on every platform.

Split Execution is Hard (Cont.)

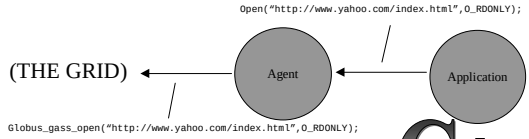
- The code replacing system calls must be able to execute the original system calls!
- Example: Sandboxing
  - Trap open().
  - Check for unauthorized file names.
    - Return failure for some.
    - Re-invoke the original open() for others.

Bypass Makes it Easy!



Example: GASS Agent

- Let's create an Agent that changes all calls to UNIX open() and close() into their analogues in Globus GASS. This will instrument the application with remote file fetching and staging.

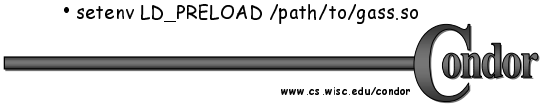


Example: GASS Agent (Cont.)

```
agent_prologue
{{
      @include "globus_common.h"
      @include "globus_gass_file.h"
}};

int open( const char *name, int flags, [int mode] )
      agent_action
      {{
            globus_module_activate( GLOBUS_GASS_FILE_MODULE );
            return globus_gass_open( namame, flags, mode );
      }};

int close( int fd )
      agent_action
      {{
            return globus_gass_close( fd );
      }};
```

Example: GASS Agent (Cont.)

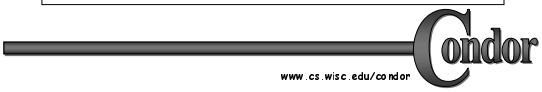
- Generate the source code.
  - bypass -agent gass.bypass
- Compile into a shared library.
  - g++ gass_agent.C (libraries) -shared -o gass.so
- Insert the library into your environment.
  - setenv LD_PRELOAD /path/to/gass.so



Example: GASS Agent (Cont.)

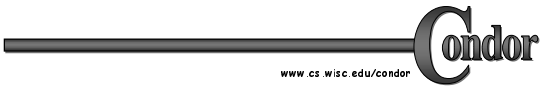
- Now, run any plain old UNIX program. The program may be given URLs in place of filenames. Globus GASS will stage and cache the needed files.

```
% cp http://www.yahoo.com/index.html /tmp/yahoo.html
% grep address http://www.cs.wisc.edu/index.html
  <LI> <A HREF="/academic.html">Academic information</A>
```



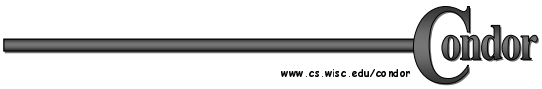
Bypass

- Uses ideas from Condor, but is a separate tool.
- User specifies design, Bypass provides details.



Bypass (Cont.)

- Can be applied to any unmodified, dynamically-linked UNIX program at run time.
  - Works on Linux, Solaris, IRIX, OSF/1.
  - Static linking only on HP-UX.



Bypass (Cont.)

- The "knowledge file" is certainly not complete!
  - Our experience: Each new OS version has new tricks in the standard library that must be foleded into the knowledge file.

Outline

- Introduction
- Using Remote I/O
- Under the Hood
- Build Your Own: Bypass
- **Conclusion**

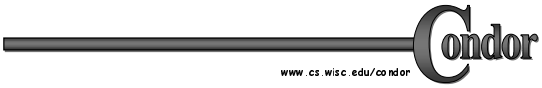
Future Work

- Lots of new plumbing, but still adding faucets
  - FTP, SRB, GASS, SAM …
- Find and use third-party staging grounds?
  - Turn checkpoint server into general staging ground.

www.cs.wisc.edu/condor

Future Work (Cont.)

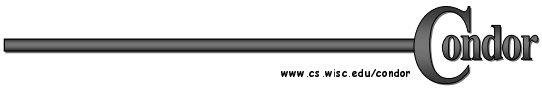
- Interaction with CPU scheduling:
  - Release CPU while waiting for slow tape?
  - Stage data, then allocate CPU?



www.cs.wisc.edu/condor

In Summary…

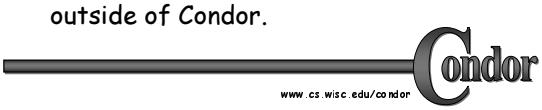
- Harnessing large numbers of CPUs requires that you use unfriendly machines.
- Remote I/O is an adapter which provides a friendly execution environment on an unfriendly machine.



www.cs.wisc.edu/condor

In Summary… (Cont.)

- Condor uses remote I/O to homogenize the many machines in a Condor pool.
- Bypass allows the quick construction of split execution systems, allowing remote I/O techniques to be used outside of Condor.



www.cs.wisc.edu/condor

Need More Info?

- Demo of Bypass on Wednesday in Room 3381.
- Contact Douglas Thain (thain@cs.wisc.edu)
- Questions now?

www.cs.wisc.edu/condor