Finding Idle Cycles
or
High Throughput Computing

Adam Bazinet and Alan Sussman
Midterm exam on Thursday, November 16
  • Sample questions posted on Exams web page
Interim report for group project due Nov. 13, 6PM
Last chance to sign up for Zaratan tour – on Wednesday, Nov. 29
Condor

- Developed at the University of Wisconsin-Madison
- Condor is aimed at High Throughput Computing (HTC) on collections of distributively owned resources
- Mainly used to scavenge idle CPU cycles from workstations (typically desktop machines and clusters)
Typical Condor Pool

- Process Spawned
- ClassAd Communication Pathway

Central Manager

- master
- startd
- negotiator
- schedd

Submit-Only

- master
- schedd

Regular Node

- master
- startd
- schedd

Execute-Only

- master
- startd

CMSC714 - Alan Sussman and Adam Bazinet
Condor Daemons

- `condor_master` - keeps other daemons running
- `condor_startd` - advertises a given resource
- `condor_starter` - spawns a remote Condor job
- `condor_schedd` - local job scheduler
- `condor_shadow` - coordinates with submitted job
- `condor_collector` - keeps status of Condor pool
- `condor_negotiator` - does all matchmaking
Condor Universes

- Universes are runtime environments for jobs
  - **Standard** universe
    - Provides checkpointing and remote system calls
    - Application must be re-linked with \texttt{condor\_compile}
  - **Vanilla** universe
    - Instead of with remote system calls, files are accessed with NFS/AFS or explicitly transferred to the executing host
- Other universes: PVM, MPI, Globus, Java, Scheduler

CMSC714 - Alan Sussman and Adam Bazinet
Matchmaking

- Matchmaking is Condor’s scheduling mechanism
- Jobs specify their requirements as a list of attributes and values
- Resources advertise their capabilities as a list of attributes and values (ClassAds)
- The `condor_negotiator` matches jobs to resources using these criteria
Condor - A Hunter of Idle Workstations

Michael J. Litzkow, Miron Livny, Matt W. Mutka
Previous Work

• In three key areas:
  • The analysis of workstation usage patterns
  • The design of remote capacity allocation algorithms
  • The development of remote execution facilities
Design Goals

• Condor is designed to serve users executing long running background jobs on idle workstations

• Job placement should be transparent

• Job migration should be supported

• Fair access to cycles is expected

• The system should be low overhead
The Scheduling Spectrum

- At one end: a centralized, static coordinator would handle scheduling
- At the other end: workstations cooperate to conduct a scheduling policy
- In the middle: Condor!

Figure 1: The Condor Scheduling Structure.
Remote Unix (RU) Facility

- Turns idle workstations into cycle servers
- When invoked, a *shadow* process runs locally as the surrogate of the remotely executing process
- System calls go over the network back to the *shadow* (an RPC of sorts)
- Used in the *standard* universe, nowadays
Checkpointing

- When a job is interrupted, RU checkpoints it - the state of the program is sent back to submitting machine, and the job may be rescheduled.

- Checkpoints consist of the text, data, bss, and stack program segments, registers, status of open files, outstanding messages to the shadow, and so on ...

- So to restart the job has to run on on a compatible system.
Checkpointing (cont’d)

- Adding checkpointing requires re-linking an application with `condor_compile`, which fattens up the binary a good deal.

- Programs now use much more RAM than they did in the past, so checkpointing in the Condor fashion may be problematic in some (many?) cases...
Fair Access to Remote Cycles

- By means of the Up-Down algorithm
- In essence, the fewer cycles you burn, the greater your priority over other users of the system... (a dynamic equilibrium)

```
pknu77@leucine:~
> condor_userprio
Last Priority Update: 11/17 23:33

<table>
<thead>
<tr>
<th>User Name</th>
<th>Effective Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:cerca@umiacs.umd.edu">cerca@umiacs.umd.edu</a></td>
<td>0.99</td>
</tr>
<tr>
<td><a href="mailto:austinjp@umiacs.umd.edu">austinjp@umiacs.umd.edu</a></td>
<td>69.91</td>
</tr>
<tr>
<td><a href="mailto:freed@umiacs.umd.edu">freed@umiacs.umd.edu</a></td>
<td>143.34</td>
</tr>
</tbody>
</table>

Number of users shown: 3
```
Performance Study

- 23 workstations executing Condor jobs were monitored for 1 month
- Study simulated a “heavy” user, and several light users
- Jobs ranged from 30 minutes to 6 hours
- Queue length as high as 40 jobs, for the heavy user
Results

- On average, light users didn’t have to wait long for their jobs to run - that’s good
- Utilization of remote resources was substantially increased - an additional 200 machine days of capacity were consumed by the Condor system
- Coordinator predicted to be able to manage at least 100 workstations with low overhead
Results (cont’d)

• Average cost of job placement and checkpointing was 2.5 seconds (again, would be higher nowadays)

• On average, all jobs experienced less than one checkpoint per hour

• Remote Unix calls are 20x more expensive than a comparable local call

  • A metric called leverage is defined as the ratio of remote capacity consumed to local capacity consumed
Conclusions

• The major design goals were achieved!
  • Job placement is transparent
  • Job migration is supported
  • Fair access to cycles is granted
  • The system is low overhead
Condor Today

- Condor has been extremely successful
- It is used by a variety of organizations: large corporations, small businesses, and of course, academic institutions
- At least one company formed to provide Condor support: www.cyclecomputing.com
- And now it is called HTCondor
Top Five Myths About Condor

- **Myth**: Condor requires users to recompile their applications.
  - **Reality**: Condor runs ordinary, unmodified applications.

- **Myth**: Condor has a single point of failure.
  - **Reality**: Condor has excellent failure isolation.

- **Myth**: Condor is only good at "cycle stealing."
  - **Reality**: Condor can effectively manage many kinds of distributed systems.

- **Myth**: Condor only runs sequential jobs.
  - **Reality**: Condor has extensive support for parallel programming environments.
Designing a Runtime System for Volunteer Computing

David P. Anderson, Carl Christensen, Bruce Allen
BOINC

• BOINC - Berkeley Open Infrastructure for Network Computing
• A platform for volunteer computing
• Popular in the scientific community
• Well established projects include SETI@home, Folding@home, LHC@home, and about 30 others currently
Design Goals

- To attract and retain volunteers
- To handle widely varying applications
- Support for application debugging
- Support for all popular platforms
BOINC Server

• One per project

• Hands out work to clients

• Keeps track of work to be done for a specific application, available hosts, state of jobs currently running, and where output files end up – all in an RDBMS

• Uses lots of threads to keep everything going w/o much overhead

• Uses *adaptive replication* to make sure all jobs get done in a timely way, even with unreliable clients
BOINC Runtime System

- Consists of an application, the core client, the BOINC manager, and an optional BOINC screensaver
BOINC Core Client (CC)

- Can be run as a standalone command line program, or as a service
- Responsible for scheduling applications
- Also checks resource consumption of the running application
- BOINC runtime library allows application to interact with core client
Architecture: Shared Memory

- For each application, the CC creates a shared memory segment containing a number of unidirectional message channels.
Applications are threaded (pthreads on UNIX, native threads on Windows)
Compound Applications

- Consists of several programs - typically a coordinator that executes one or more worker programs (so a workflow)
Task Control

- CC can perform various operations on running tasks: suspend, resume, quit, abort
- These operations are implemented by sending messages to the process control channel
Status Reporting

- CC needs to know the CPU time and memory usage of each application every second (or so)
- The BOINC runtime library makes the measurements and reports them through the status channel
Credit Reporting

- By default, credit is computed by multiplying a benchmark score by the application’s total CPU time.
- However, for a number of reasons, this estimate can be erroneous.
- Hence, there is support in the BOINC API for allowing the application to directly compute floating point operations.
BOINC must run tasks in separate directories, but want to avoid making unnecessary copies of data

- `boinc_resolve_filename("infile", physical_name);`
- `f = boinc_fopen(physical_name, "r");`
Checkpointing

- Not absolutely necessary, but extremely helpful when trying to get long-running results back, or when a reliable turnaround time is desired.
- Checkpointing scheme is application specific! Unlike the Condor mechanism...
- BOINC users care about checkpointing immensely (and will harass you indefinitely until you implement it).
Graphics

- Applications supplied graphics are viewable either as a screensaver or in a window.
- BOINC runtime library limits the fraction of CPU time used by the graphics thread.
Remote Diagnostics

- Application’s standard error is directed to a file and returned to the server for all tasks.
- If an application crashes or is aborted, a stack trace is written to standard error.
- Problems may occur only with specific OSes, architectures, library versions, etc.
Long-running Applications

- Some projects run tasks that take an extremely long time to complete.

- Besides checkpointing, other mechanisms are necessary to support these tasks - for example, periodically granting users credit, or communicating intermediate results to the server for processing.

- These mechanisms use the trickle messages channel.
Conclusions

• BOINC is very flexible - it satisfies those who want it to stay out of the way completely, as well as those who really want to be involved in the science

• BOINC supports a wide range of applications and runs on every major platform

• Current version includes using GPUs and multicore machines (and run multithreaded applications)