Projects
UC Berkeley developers (2.5)
Computer scientists
PC volunteers (300,000)
Other volunteers: testing translation support
Workshop goals

- Learn what everyone else is doing
- Form collaborations
- Steer BOINC development
  - tell us what you want
Hackfest (tomorrow)

- Goal: get something concrete done
  - Improve docs
  - design and/or implement software
  - learn and use a new feature
The state of volunteer computing

- Volunteers: stagnant
  - BOINC: 290K people, 450K computers
- Science projects: stagnant
- Computer science research: stagnant
- Let’s keep trying anyway
Requests to projects

• Do outreach
  - notices
  - automated emails
  - mass emails
  - message boards
  - mass media

• Use current server code
To developers/researchers

- Talk with me before starting anything, especially if it’s of general utility

davea@ssl.berkeley.edu
What’s new in BOINC?

- Storage and data-intensive computing
- Virtual machine apps
- GPU apps
- Scheduling
- Remote job submission
- Other
Storage and data-intensive computing

- Disk space
  - average 50 GB available per client
  - 35 Petabytes total
- Trends
  - disk sizes increasing exponentially, faster than processors
  - 1 TB * 1M clients = 1 Exabyte
BOINC storage architecture

Applications

- Data archival
- Result archival
- Local scheduling
- Dataset storage

BOINC storage infrastructure
BOINC storage infrastructure: managing client space

- Volunteer prefs determines BOINC's allocation
- Allocation to projects is based on resource share
BOINC storage infrastructure: RPC/server structure

- “Sticky file” mechanism
Volunteer data archival

- Files originate on server
- Chunks of files are stored on clients
- Files can be reconstructed on server (with high latency)
- Goals:
  - arbitrarily high reliability (99.999)
  - support large files
Replication

- Divide file into $N$ chunks
- Store each chunk on $M$ clients
- If a client fails
  - upload another replica to server
  - download to a new client
- Problems
  - high space overhead
Erasure Coding

- A way of dividing a file into $N+K$ chunks

$N = 4$  $K = 2$

- The original file can be reconstructed from any $N$ of these chunks.

- Example: $N=40$, $K=20$
  - can tolerate simultaneous failure of 20 clients
  - space overhead is only 50%
Problems with erasure coding

- When any chunk fails, need to upload all other chunks to server
- High network load at server
- High transient disk usage at server
Two-level coding

- Can tolerate $K^2$ client failures
- Space overhead: 125%
Two-level coding + replication

- Most recoveries involve only 1 chunk
- Space overhead: 250%

$M = 2$
VDA Implementation

- **DB tables**
  - vda_file
  - vda_chunk_host
- **Scheduler plugin**
  - handle transfers, sticky file list
- **VDA daemon**
  - process files needing update, dead hosts
- **Emulator**
  - compute performance metrics
Support for large files

- Restartable download of compressed files
  - include `<gzip/>` in `<file_info>`
  - currently only for app version files
- Combine uncompress, verify
- Asynchronous file copy, uncompress/verify
  - 10MB threshold
- Handle > 2GB files; use `stat64()`
VM app support

- BOINC client
- vbox wrapper
- shared dir
- virtual machine
- Vbox service
VM app support

- Use Vbox “snapshot” mechanism for checkpointing
- Report non-ancestral PID (VM) to client
- Report network traffic to client
- Use Remote Desktop Protocol to allow user to view console
- CPU throttling
- Multicore
GPU app support

- Pass device type and number in init_data.xml
- OpenCL initialization: boinc_get_opencl_ids()
- Plan classes configurable in XML file
Scheduling: batch-level (proposed)

- Policy: feeder enumeration order
- Goals
  - Give short batches priority over long batches
  - But don’t let a stream of short batches starve long batches
  - enforce user quotas over long term
Scheduling: batch-level

- Each user has “logical start time” $LST(U)$
  - when submit batch, increment by expected runtime / share($U$)
- Each batch has “logical end time” $LET(B)$
  - set to $LST(U) +$ expected runtime
- Give priority to batch for which $LET(B)$ is least
Scheduling: job-level

- **Policies**
  - feeder enumeration order
  - job selection from shared mem cache
  - choice of app version
  - deadline assignment

- **QoS types**
  - non-batch, throughput-oriented
  - Long-deadline batches
  - As fast as possible (AFAP) batches
  - short-deadline batches
Scheduling: job-level

• Goals
  – accelerate batch completion
  – avoid tight job deadlines
  – avoid long delays between instances
  – minimize server configuration
Scheduling: job-level

- For each (host, app version) maintain percentile incorporating
  - average turnaround time
  - consecutive valid results
- Dynamic batch completion estimation
  - based on completed and validated jobs
Scheduling: job-level

- Feeder enumeration order
  - LET(J) ascending, # retries descending
Scheduling: job-level

For each job
  - for each usable app version AV
    - if $x < \text{est\_completion}(B)$
      - send job using AV with deadline $\text{est\_completion}(B)$
    - else if $\text{percentile}(H, AV) > 90\%$
      - send job using AV with deadline $x$
Locality scheduling

- Have a large dataset
- Each file in the dataset is input for a large number of jobs
- Goal: process the dataset using the least network traffic
- Example: Einstein@home analysis of LIGO gravity-wave detector data
Locality scheduling

- Processing jobs sequentially is pessimal
  - every file gets sent to every client
Locality scheduling: ideal

- Each file is downloaded to 1 host
- Problems
  - Typically need job replication
  - Widely variable host throughput
Locality scheduling: proposed

teams

jobs

jobs

jobs

jobs
Locality Scheduling Lite

- Optional feature of existing scheduler
- Use when \# files < \# shared-mem slots
Remote job submission

- Operations
  - estimate, submit, query, abort, get result files, retire
Remote job submission

- Input file options
  - local: file already exists on server
  - inline: file is passed in request XML
  - semilocal: file is accessible via HTTP from server; server fetches and serves it
  - remote: file is on a server accessible to clients; must supply size and MD5
Broadcast and targeted jobs

- Broadcast jobs
  - run once on all hosts, present and future
  - can limit to user or team
  - Not handled by validator or assimilator

- Targeted jobs
  - targeted to a host, user, or team
  - handled by validator, assimilator
  - can do this when create job, or dynamically
Git migration

• Branches
  – master (development)
    • new code goes here
  – server_stable
    • hot fixes may go here
  – client_release_X_Y
    • hot fixes may go here
Other things for CERN T4T

- Web-based app graphics
  - app implements an HTTP server
  - port is conveyed to Manager
  - “app graphics” opens a browser window
- “need network” app version flag
  - don’t run if network not available
New OS support

- Windows 8
- Mac OS X 10.8
  - Xcode 4.5
- Debian 6.0
- Android
Large DB IDs

- SETI@home has done > 2B jobs
- made IDs unsigned (31->32 bits)
- eventually will need to move to 64 bit
Validator

- Runtime outlier flag
  - don’t use this job in runtime, credit statistics
- Test harness
  - validator_test file1 file2
BOINC in app stores

- Operated by OS vendors (Apple, MS, Google)
- Vendor screen apps and takes a cut
- Goal: package BOINC for app stores
  - and maybe project-specific versions
Didn’t get done

- OpenID support
- remodel computing preferences