SETI@home and BOINC

Alan Sussman

CMSC 818S

March 6, 2007

Notes

- Project issues?
- Next lecture will be led by Malina
  - on enterprise desktop grids
  - but first, talk about project results
- Next week more on desktop grids, etc.
  - me Tuesday, Gary Thursday
- Following week is spring break
  - then I'm out of town for a week at a conference
  - Jik-Soo Kim will lead Tuesday, 3/27, on peer-to-peer our desktop grid project
  - what about Thursday?

SETI@home

Public resource computing

- Now often called desktop grids
- Works best for problems with high compute to data ratio, and on problems with independent tasks
  - also good if there are methods to tolerate errors
- Many Internet-connected computers used to analyze radio telescope signals, to detect signs of intelligent life outside Earth (SETI)
  - narrow bandwidth radio signals from space likely produced by extraterrestrial technology
  - data source is Arecibo radio telescope
  - initially poor Internet connection (56Kbps), so data transmitted to Berkeley lab on 35GB DLTs

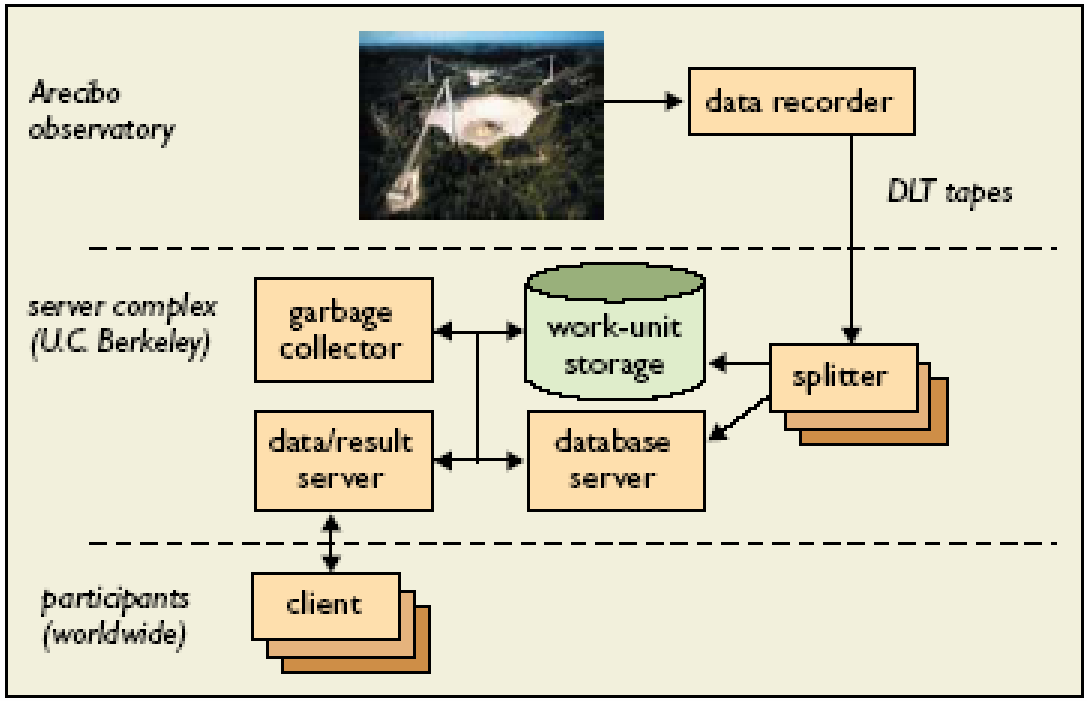
Basic design

- Server-client architecture
- Server determines work units from data acquired, hands out to clients as they request it (when free)
- Server assigns work unit multiple times
  - for dealing with faults and malicious clients
  - store results and current status of all work units in a commercial relational DB
  - this is the part that fails regularly, and does not scale well
- Client gets work unit from server, analyzes it, returns result to server
  - only communicates to get work unit, return result, can be otherwise disconnected from network
  - only computes when host is idle or in background at low priority
  - checkpoints to disk periodically

More on client

- Multithreaded C++ program
  - runs on many platforms, using GNU tools (gcc, autoconf)
  - one thread for communication and data processing
  - one thread for GUI interactions (GUI optional)
  - one thread for graphics rendering (screensaver)
  - communication between threads via shared memory data structures

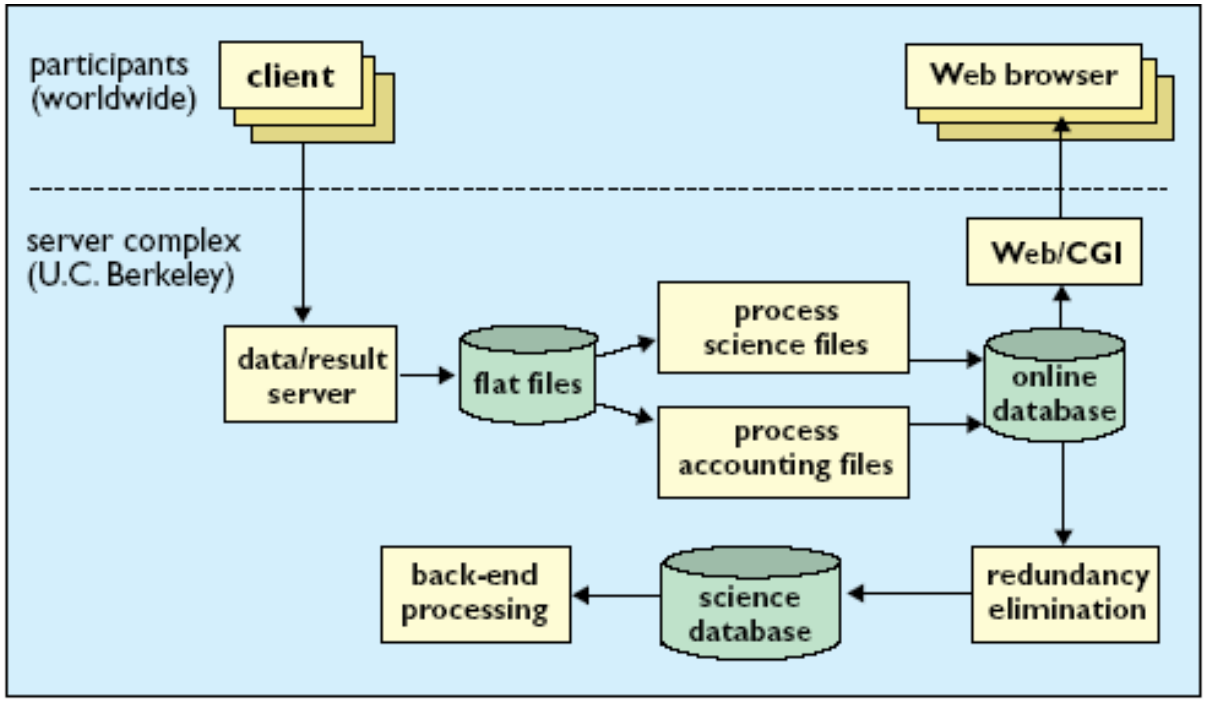
Getting the data to clients



More on server

- First write client result to a disk file
- Another program reads the files, and stores the data (result and signal records) into database
- Also write a log entry in a file to give client “credit”
  - keep track of users, teams, countries, CPU types
  - log entries flushed to database periodically
- Final part of data analysis done at server
  - remove redundant results via voting algorithm
  - identify and remove man-made signals
  - final manual check on “interesting” signals, which can then be re-observed (look at the input data again)

Collecting data from clients



Statistics

- 200K initial clients in May 1999, over 3.9M by August 2002
- In 1 year starting 7/01, clients processed 221M work units, with average throughput of over 27TFlops
- Total computation up to some point in time is 1.87×10^{21} floating point operations
  - claim is it's the largest computation on record

Other issues

- Publicity is very important to them
  - users competing to complete most work units
- Web server has been compromised, and hackers have obtained email addresses from server (bug in client/server protocol)
  - cheating to get credits is also an issue
- Misbehaving and malicious users (clients)
  - includes modifying executable to run faster on specific processors – issue is correctness of results
  - sending erroneous results – mostly taken care of through redundancy, other methods can be used too

BOINC

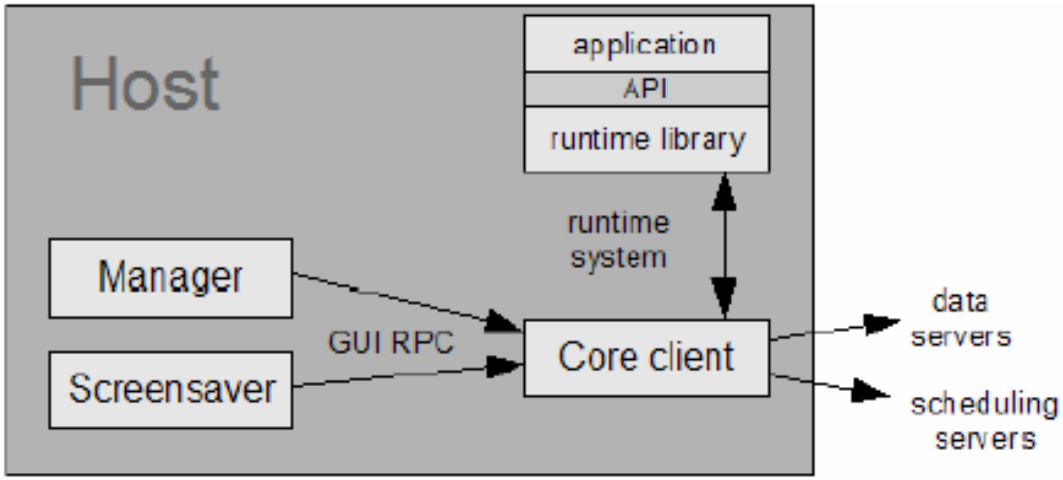
Overview

- Middleware system (API and library) for building client/server desktop grid system for an application
- Application project provides servers, clients provided by volunteers
- BOINC has server side components (schedulers, demons for distributing tasks, collecting results, etc.), but the paper is about the client side components

BOINC goals

- Like Condor, client hosts should not be abused, unless they want to be
  - user can specify preferences – e.g., when the client can run, in background or only when idle, etc.
- Volunteers get incentives to participate
  - graphics/screensaver – application-specific
  - competition to provide lots of CPU hours – need web sites, and good statistics collection
- Autonomic software – easy to install, maintain, and use (meaning set preferences)
- Handle widely varying applications and task granularity
- Also want to provide debugging/diagnostic info back to project developers, and support many platforms

BOINC architecture

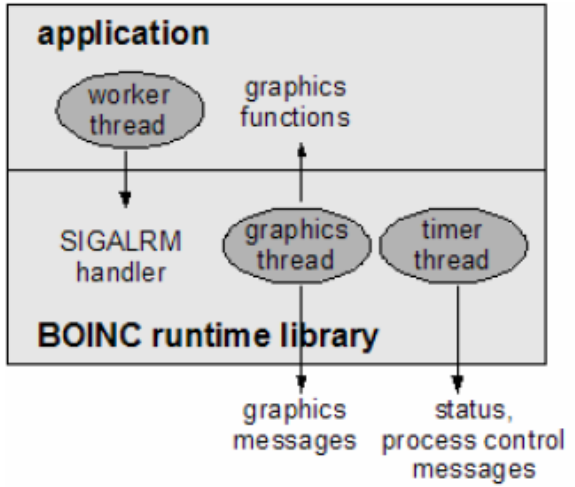


- Core client communicates with schedulers, uploads/downloads files, executes/coordinates apps
  - can schedule multiple apps on a multiprocessor host
  - preemptive round robin scheduling among apps, for client with multiple apps
  - runtime system here, runtime library linked into apps
- Manager has GUI to allow users to view and control computation status – communicates with core client via RPC
- Screensaver (if enabled) runs when host idle, gets graphics from application

BOINC communication

- Shared-memory message passing between core client and apps
- 8 uni-directional message channels, 4 each way
  - task control (suspend, resume, quit, abort), graphics, status, trickle messages
  - each a fixed size buffer and a full/empty bit
  - messages in XML

BOINC thread structure



- Worker thread executes app, does accounting, suspends app if needed
- Graphics thread deals with GUI, calls app rendering function
- Timer thread executes timer function once per second (to handle process control messages)

Task management

- To suspend/resume worker thread (the app)
- To quit the app, and restart later
- To abort the app, with no restart later
- All implemented via messages to process control channel
- Core client can force quit/abort if the app does not deal with the message
- Heartbeat messages from core client to app once per second, app exits after 30 seconds w/o a heartbeat message

Accounting

- Core client keeps track of CPU and memory usage by each app, once per second
- Also need estimate of how close app is to completion, for scheduler and GUI
  - app calls boinc_fraction_done(double) to tell client
- Keep track of how much work done by each host and each volunteer at server, also displayed in manager GUI
  - use BOINC API op count functions if default counts inaccurate for app

Dealing with App Data

- Each task runs in a separate directory
  - read-only files shared via symbolic links (not exactly, but close enough)
- Checkpointing
  - user preference set to determine minimum interval between disk activity
  - app calls bool boinc_time_to_checkpoint() to say app is in checkpointable state
    - returns true if minumum interval has elapsed
    - then app should write checkpoint file and call boinc_checkpoint_completed()
  - while checkpointing is going on, quit messages are disabled
  - also have to be careful about partially written output files
    - BOINC has printf() replacements that buffer in memory and flush on checkpoint

Remote diagnostics/debugging

- App's stderr directed to file, returned to project server for all tasks (not just failed ones)
- If app crashes, stack trace written to stderr
  - if executable has a symbol table, trace uses symbolic names
- If app aborts (task exceeded time, disk or memory limits, or user aborted it), stack trace written to stderr
- All this info is stored in server DB

Long running applications

- Trickle messages
  - App sends messages to server with accounting info (so doesn't wait until end of task)
  - App sends server messages with summary of computational state – so server can kill if necessary
  - Server sends app message to terminate
- File uploads
  - BOINC API has functions to tell core client that an output file is complete and should be uploaded, instead of waiting until task is complete

Limitations/plans

- Limited goals
  - no binary compatibility (separate app executable per platform)
  - no sandboxing – just limits on CPU, memory, disk usage
    - plan to have apps run as an unprivileged user
  - no process migration – replicate computations instead
- Plan to support graphics coprocessors as a schedulable resource (like another CPU)
- Plan to support desktop grid within an organization –what does that mean?