Volunteer Computing - Scheduling in BOINC

Ryan Stern
stern@cs.colostate.edu

Department of Computer Science
Colorado State University

BOINC

- The Berkley Open Infrastructure for Network Computing
- A middleware for volunteer computing
  - Sometimes called public-resource computing
- Utilizes the computing resources of hundreds of thousands of volunteers

Challenges of Volunteer Computing

- The resources are heterogeneous
  - In terms of size and speed
  - In terms of availability and reliability
  - In terms of operating system and architecture
- The resources are not under a single administration
  - Have limited control over when a job executes
  - Can't vouch for the security of the resources
  - Can't control existing firewalls
- Need to encourage and retain volunteers

BOINC Challenge:
Machines aren’t under project’s control

- Participants may not complete the work given
  - Or complete the work, but with invalid results
- Solution: Redundant computing
  - Other participants pick up abandoned work
  - Ensures correctness of results using a quorum
  - Finds nodes that constantly give bad results (malicious or broken)
- Most participants have a firewall installed
- Solution: Clients must request work
  - Scheduling servers do not contact clients

BOINC Challenge:
Resource availability

- Resources can become unavailable at any time
- Every so often, tasks save a checkpoint
- Checkpoints are used to resume processing if the machine is shut down
**BOINC Challenge: Network availability**

- Not all participants have a continuous network connection
- Solution: Download and queue enough work to bridge the gap between internet connectivity
- Some participants may have bandwidth limits
- Solution: Send jobs with smaller inputs to participants that have bandwidth limits
- Want to ensure that client requests don’t overload the servers
- Solution: Defer requests by a certain amount of time
  - When unable to connect to a server, exponentially increase the wait

**BOINC Challenge: Retaining Volunteers**

- Participants need to feel like their contributions are worthwhile
- Solution: Award credit for work done
  - Participants compete against each other
  - Participants can join and contribute to a team
- BOINC projects have graphics showing computation progress
  - Graphics are designed to be used as screensavers

**BOINC Challenge: Heterogeneous resources**

- Implement the concept of a workunit
  - Includes the application and its inputs
  - The processing, memory, and storage requirements
- Having a variety of application types improves performance
- Supports homogeneous redundancy
  - For applications that produce architecture-specific results
- Need a client side scheduler and a server-side scheduler

**What’s the problem?**

- An optimization problem
- We want to reduce time spent idle without work
  - While attempting to reduce lateness and waste
  - All while respecting client resource shares and other settings

**Three Approaches to Improve Scheduling**

1. Selecting best client policies
2. Evaluation of server policies (EmBOINC)
3. Evaluation of threshold polices
Solution One
Selecting Client Scheduling Policies

Policies evaluated

- **CPU scheduling policies**
  - CS1: round-robin time-slicing
  - CS2: simulate round-robin scheduling and identify jobs that miss their deadline; schedule these jobs earliest deadline first

- **Work fetch policies**
  - WF1: fetch enough work for buffer_min_days + buffer_additional
  - WF2: maintain debt owed to each project; simulate round-robin scheduling and identify CPU shortfall; fetch jobs for the project with the highest debt - shortfall.

- **Work send policies:**
  - WS1: When requesting X seconds of work, send at least X seconds
  - WS2: Request includes jobs already assigned with the deadlines and completion times; simulate earliest deadline first with candidate jobs added; only send jobs that don’t worsen missed deadlines

- **Job completion estimation policies:**
  - JC1: Based on FA + (1 - F)B; F is fraction done, A is estimate based on elapsed CPU time, B is estimate based on benchmarks and preferences
  - JC2: Maintain a per-project duration correction factor

Scheduling Scenarios

- Simulated client behavior implemented by refactoring the BOINC client source code
  - Responses from the project servers are simulated

- The client and project configuration used are based on data collected from real BOINC projects

Scheduling Scenarios: Metrics

- Idleness
  - The fraction of available CPU time that is unused due to a lack of jobs

- Waste
  - The fraction of CPU time used belonging to jobs that miss the deadline

- Share violation
  - Measure of how closely resource shares are respected

- Monotony
  - Inverse measure of how often the host switches between projects

Scheduling Scenarios: CS Results

10/1/2013
Scheduling Scenarios: CS Results

Scheduling Scenarios: WF/WS Results

Scheduling Scenarios: JC Results

Solution Two Evaluation with EmBOINC

EmBOINC

- A tool for evaluating project configurations

EmBOINC Simulator

- Uses a discrete event simulator to model user behavior
- The server behavior was determined to be too complex and tedious to implement with a simulator
EmBOINC Emulator

- Interacts with existing BOINC servers
  - Reduces the amount of maintenance required between versions
- Introduces a challenge: Two different time systems
  - The events happen in simulated time
  - The BOINC daemons expect wall-time to function properly
- Solution: Have all BOINC daemons synchronize the time after receiving a signal

Host Modeling

- Utilizes host information taken from a real project database
- Probabilistic attributes are estimated statistically
  \[
  \text{pdf}(x; \alpha, \beta) = \left( \frac{x}{\beta} \right)^{\alpha - 1} e^{-x/\beta} \]

Job Characterization

- For each application, EmBOINC allows the user to specify:
  - The length of the job
  - Number of replicas and quorum
  - The available platforms and required hardware
  - The sensitivity to operating systems and architecture
- This information can also be extracted from an existing project’s database

EmBOINC Results

Motivation

- Current BOINC scheduling is first-come-first-serve
  - Generate jobs and distribute to first compatible machine
- This solution aims to improve the rate of valid results and the speed at which they are returned to the server
SimBA

- Simulator of BOINC Applications
- Another discrete event simulator
- Simulates the BOINC master services
- Utilizes traces from real BOINC projects to model the behaviors of participants

Metrics

- Availability
  \[ \text{availability}(t) = \frac{\text{WU completed}(t)}{\text{WU distributed}(t)} \]
- Reliability
  \[ \text{reliability}(t) = \frac{\text{WU validated}(t)}{\text{WU completed}(t)} \]

Policies

- First-come-first-serve policy
  - The current policy
  - Assigns the first available task, usually the oldest
  - Has a higher rate of distribution
- Threshold-based policy
  - Only assign tasks if the worker's availability and reliability are above a configurable thresholds
  - Currently permanently starves workers that fall below the threshold

Starvation Issue

- Assigning new tasks to unreliable workers is not desirable
- Potential solution: Sliding window
  - Set an expiration date on records indicating poor performance
  - Leaves a worker idle until the score improves
- Potential solution: Redundant tasks
  - Generate additional redundant tasks
  - Project does not depend on tasks completing successfully
  - Arguably wasting processing time, but result may arrive sooner

Discrete Event Simulator

- Entities
  - Workunit generator
  - Worker generator
  - Worker
  - Workunit
  - Workunit instance (task)
- Events
  - Generate workunit
  - Generate worker
  - Request tasks
  - Generate tasks
  - Compute task output

<table>
<thead>
<tr>
<th>SimBA Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siml.DURATION (hours)</td>
<td>100</td>
</tr>
<tr>
<td>SIM. INTERVAL (hours)</td>
<td>30</td>
</tr>
<tr>
<td>SIM. MODEL. TIME.SIZE</td>
<td>600</td>
</tr>
<tr>
<td>MAX. TASK. PER. WORKER</td>
<td>12</td>
</tr>
<tr>
<td>TASK. DEP. TIME. (min)</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 2: SimBA setup.
SimBA Results

Commonalities & Contrasts

Commonalities

• System complexity is an issue
• Two approaches used the existing source code
• Two approached built a discrete event simulator

Contrasts

• The approaches target different parts of the system
  – 1st: Looked mostly at client-side scheduling
  – 2nd: Looked at evaluating server-side scheduling
  – 3rd: Looked at limiting workers with poor results

• Metrics and goals are different
  – 1st: Idleness, waste, share violations, monotony
  – 2nd: valid rate, latency, idleness
  – 3rd: success rate, valid rate