Volunteer Computing - Scheduling in BOINC

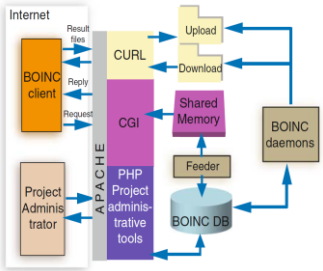
Ryan Stern
stern@cs.colostate.edu

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

BOINC



Challenges of Volunteer Computing

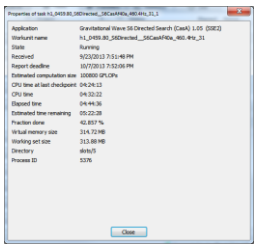
- The resources are heterogeneous
  - In terms of size and speed
  - In terms of availability and reliability
  - In terms of operating system and architecture
- The resources are not under a single administration
  - Have limited control over when a job executes
  - Can't vouch for the security of the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

- Recourses 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

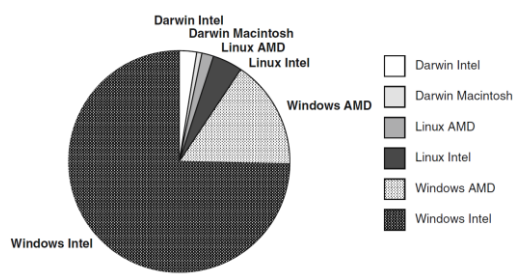
BOINC Challenge: Heterogeneous resources

- 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

BOINC Challenge: Heterogeneous resources



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.

Policies evaluated

- 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

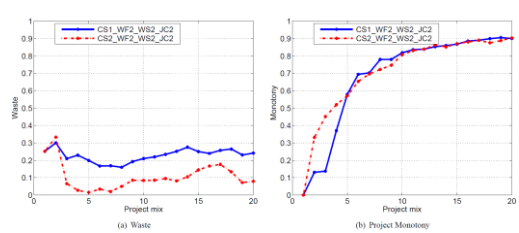
Scheduling Scenarios

- 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

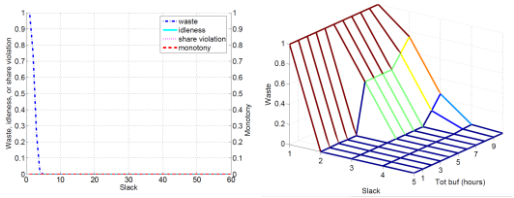
- Idleness
  - The fraction of available CPU time that is unused due to a lack of jobs
- Waste
  - The fraction of CPU time used belonging to jobs that miss the deadline
- Share violation
  - Measure of how closely resource shares are respected
- Monotony
  - Inverse measure of how often the host switches between projects

Scheduling Scenarios: CS Results

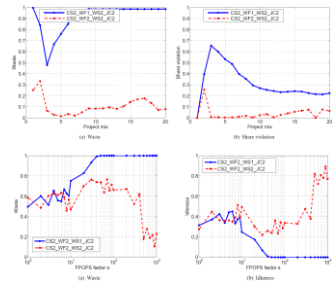


(a) Waste (b) Project Monotony

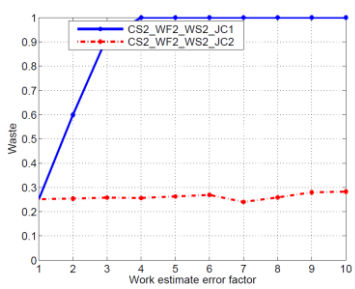
Scheduling Scenarios: CS Results



Scheduling Scenarios: WF/WS Results



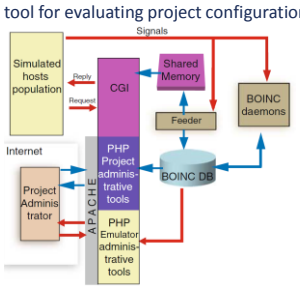
Scheduling Scenarios: JC Results



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

Table 1 Table of events triggered by entities

EmBOINC Init	**Job Generator**
1 Set next time	5 Read traces
2 Get entity's time	6 Read job
3 Enqueue	7 Make job
4 Terminate	
EmBOINC DES controller	**Host generator**
5 Read configuration	5 Read traces
6 Read traces of users	6 Read host
7 Read user	7 Make host
8 Make user	
9 Update simulated time	**Simulated host**
10 Pop entity from queue	5 Write request
11 Execute entity	6 Send request
12 Call transitioner	7 Parse reply
13 Call feeder	8 Get job
14 Call validator	9 Get job instance
15 Call assimilator	10 Execute job instance
16 Wait for signal	11 List current job load
17 Continue simulation	12 Calculate requested job

Sequence of events

EmBOINC DES controller:
5, 6, (7, 8)+, 9, (10, 9, 11, [12, 13])+, 12, 13, 14, 15, 16, 17)†, 4
Work generator: (2, 5, (6, 7)*, 1, 3)+, 4
Host generator: 2, 5, (6, 7)*, 4
Simulated host: ((2, (10, 12)∗, 11, 5, 6, 7, (8, 9)∗)∗, 1, 3)+, 4

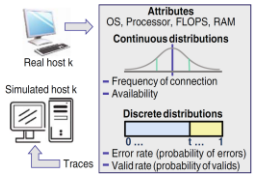
Notation: * 0 or more, + 1 or more, () group, [] optional

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

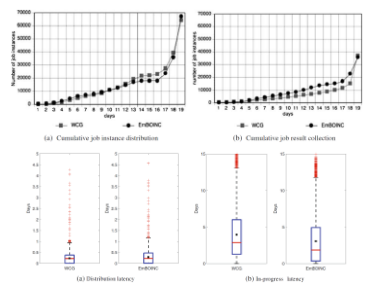


$$pdf(x;\alpha,\beta) = \left(\frac{\alpha}{\beta}\right)\left(\frac{x}{\beta}\right)^{\alpha-1} e^{-(x/\beta)^{\alpha}}$$

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



(a) Cumulative job instance distribution (b) Cumulative job result collection

(a) Distribution latency (b) In-progress latency

Solution Three
Threshold-based Scheduling

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 witch 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

- Availibility
  - $availability_i(t) = \frac{WUI_{completed\ i}(t)}{WUI_{distributed\ i}(t)}$
- Reliability
  - $reliability_i(t) = \frac{WUI_{validated\ i}(t)}{WUI_{completed\ i}(t)}$

- Work units in a bundle determines by dividing the total number of FLOPs that a worker can commit by the estimated number of floating point numbers

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

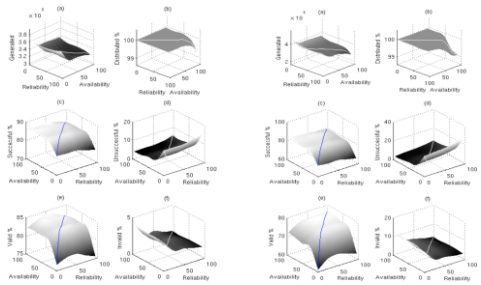
Discrete Event Simulator

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

SimBA Parameter	Value
SIM_DURATION (hours)	340
SAMPLE_INTERVAL (hours)	1
WU_INITIAL_POOL_SIZE	800
WUS_PER_HOUR	400
MAX_TASK_PER_WORKER	12
TASK_DEADLINE (hours)	48

Table 2: SimBA setup.

SimBA Results



SimBA Results

Availability/ Reliability Thresholds	Successful Results (%)	Unsuccessful Results (%)	Valid Results (%)	Invalid Results (%)
MFold				
0.95/0.95	83.59	6.65	81.94	1.64
0.75/0.75	85.58	5.40	83.71	1.88
0.50/0.50	85.47	5.96	83.18	2.29
0.25/0.25	85.64	6.52	83.21	2.43
0/0 (FCFS)	80.54	13.79	76.68	3.86
Best	86.83	4.52	84.40	1.52
Worst	77.31	15.91	75.65	4.24

Availability/ Reliability Thresholds	Successful Results (%)	Unsuccessful Results (%)	Valid Results (%)	Invalid Results (%)
CHARMM				
0.95/0.95	80.79	8.40	77.01	3.78
0.75/0.75	82.49	6.51	77.75	4.74
0.50/0.50	82.23	7.39	76.56	5.67
0.25/0.25	81.65	8.83	75.28	6.37
0/0 (FCFS)	72.19	22.21	61.26	10.93
Best	85.02	4.54	78.03	3.40
Worst	66.56	27.73	61.25	12.44

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