Scheduling for Volunteer Computing on the BOINC server infrastructure

Duco van Amstel

ENS Lyon & INRIA - Rhône-Alpes - MESCAL team

22 June 2012

Presentation Outline

1. Introduction & Research Context
2. Volunteer Computing in a Nutshell
3. Server-side scheduling architecture
4. Evaluation and Simulation
5. Conclusion

Introduction

The growth in computational has encouraged the development of new and more powerful architectures:

- Super-computers
- Computational Grids

Drawback These systems are very expensive to maintain

Idea Harness the already existing unused computer cycles

Existing projects and software for Volunteer Computing:

- Distributed.net - 1997
- SETI@Home - 1999

Research Context

SETI@Home evolved into a unified software structure:

> **B**erkeley **O**pen **I**nfrastructure for **N**etwork **C**omputing (BOINC)
> Used by multiple projects (*i.e SETI, LHC, WCG, Aids, Einstein, . . .*)

All these projects have a common point:

> $\longrightarrow$ A single dedicated server per project

Run multiple projects on the same server? (*i.e Umbrella projects*)

> $\longrightarrow$ Less infrastructure to manage
> $\longrightarrow$ More research-teams would benefit

How to run and schedule multiple projects on the same server?

Section outline

1. Introduction & Research Context
2. Volunteer Computing in a Nutshell
   - Failure Prone Environment
   - Rise of Multi-Application Projects
   - The Straggler Issue
   - Memo on Classical Scheduling
3. Server-side scheduling architecture
4. Evaluation and Simulation
5. Conclusion

Network-related Issues

Unlike a dedicated grid, the network of a Volunteer Computing platform is inherently unreliable:

1. No permanent connections (*i.e Dial-Up, DSL*)
2. One-sided connections from client to server (*Firewall, NAT*)
3. Irregular and unpredictable connection intervals

Network-related Issues

Unlike a dedicated grid, the network of a Volunteer Computing platform is inherently unreliable:

1. No permanent connections (*i.e Dial-Up, DSL*)
2. One-sided connections from client to server (*Firewall, NAT*)
3. Irregular and unpredictable connection intervals

Existing work-arounds:

Pull-mode Clients explicitly requests work from the server (1,2)

Aggregate Results of multiple tasks are aggregated before being returned (1,2)

Reconnect In case no work is available the server specifies a reconnection interval (2,3)

Client-related Issues

Clients should also be considered unreliable:

1. Cannot keep track of client's availability (*i.e holidays*)
2. Heterogeneous and exotic architectures (*i.e different floating-point, library versions*)
3. Fake results (*i.e malicious users, overclocking*)
4. Heterogeneous operating systems
5. Numerical instability of the computation

Client-related Issues

Clients should also be considered unreliable:

1. Cannot keep track of client's availability (*i.e holidays*)
2. Heterogeneous and exotic architectures (*i.e different floating-point, library versions*)
3. Fake results (*i.e malicious users, overclocking*)
4. Heterogeneous operating systems
5. Numerical instability of the computation

Again BOINC software uses some existing work-arounds:

Deadline Re-scheduling of timed-out tasks (1)

Replication Tasks are sent to multiple clients and results are compared (2,3,4)

Redundancy Replication is done on similar hosts (2,4,5)

Current BOINC status and its evolutions

Objective of the current BOINC:

Throughput maximization (nb. of tasks over time)

BOINC developers observe a diversification of the use-cases for Volunteer Computing:

- Current scheduling techniques are not adapted
- A need for more interactivity and thus response-time optimization

$\longrightarrow$ Start of the multi-application / multi-campaign context

Current BOINC status and its evolutions

Objective of the current BOINC:

Throughput maximization (nb. of tasks over time)

BOINC developers observe a diversification of the use-cases for Volunteer Computing:

- Current scheduling techniques are not adapted
- A need for more interactivity and thus response-time optimization

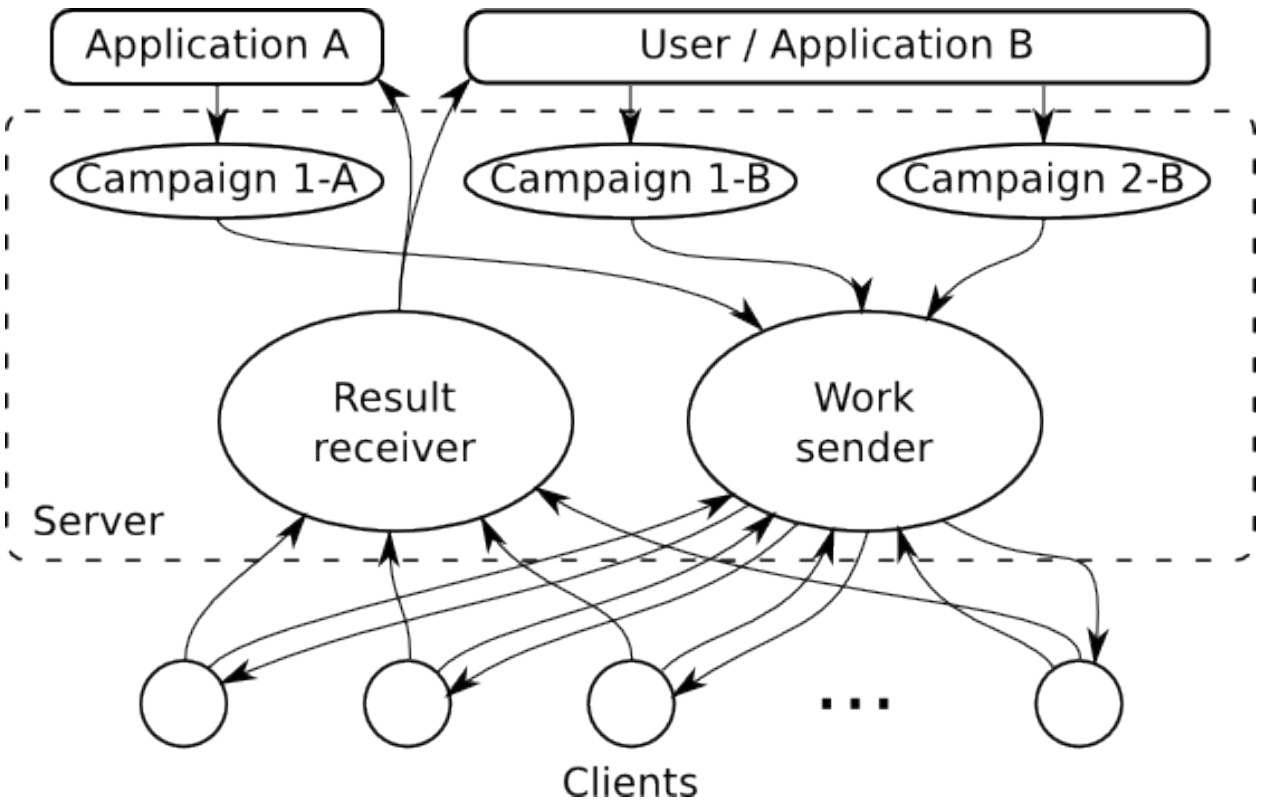
$\longrightarrow$ Start of the multi-application / multi-campaign context

Ultimate goal would be a web-based user-portal

- An interface where **users** submit jobs
- Jobs are scheduled by the BOINC server software
- Results are sent to **users** ASAP

Diverse Entities

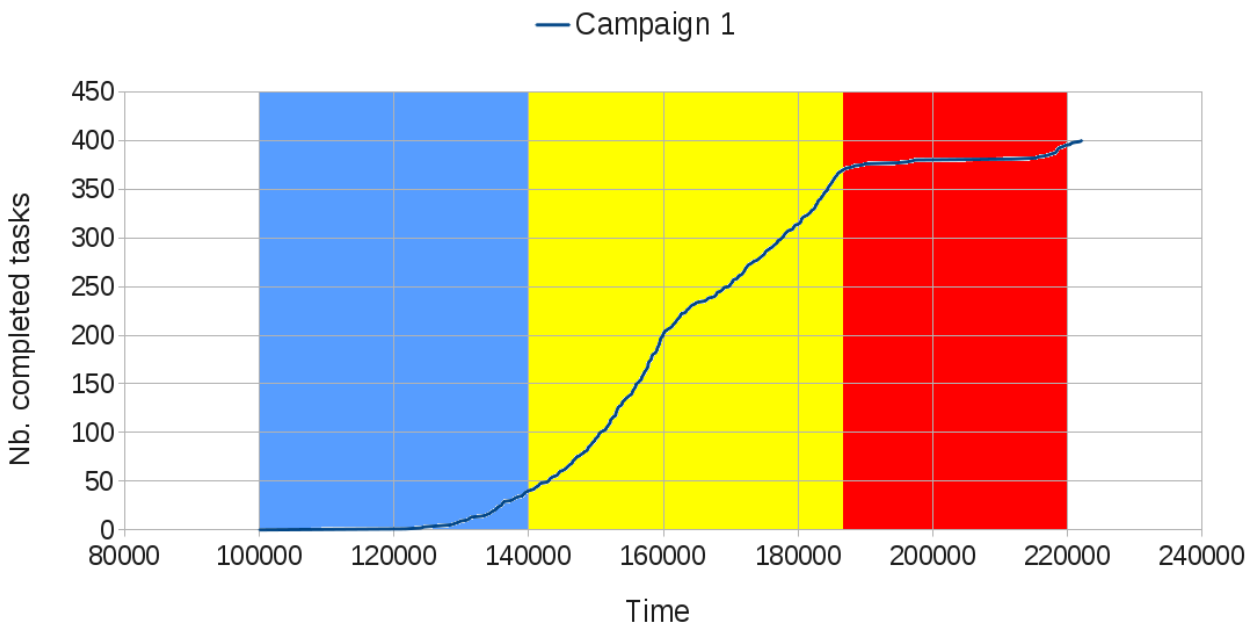
Applications (Users) / Campaigns (Jobs) / Tasks



What Stragglers Are

Individual task completion is not considered

→ Depending on hosts this may take a long time
→ Affects campaign response-time
→ Causes the appearance of "stragglers"



How to Deal with Stragglers

GridBot environment:

Volunteer Computing initiative derived from BOINC with its own specific server code

Besides using a network of volunteer hosts it also disposes of a dedicated grid infrastructure

GridBot provides some solutions to the straggler issue:

Aggressive replication

Last tasks are sent to more clients

Client selection

Last tasks are sent to reliable and dedicated clients

Adapting Deadlines

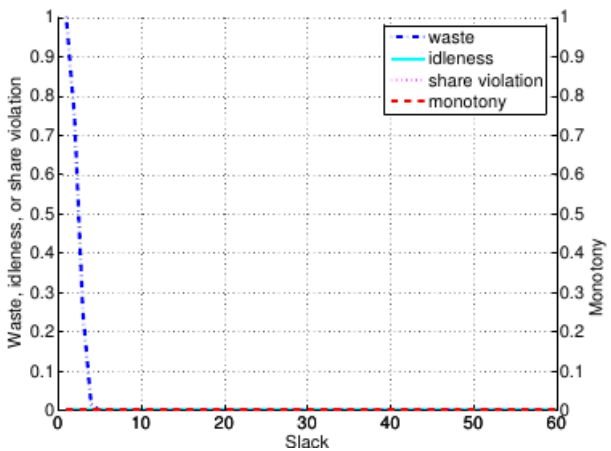
Tasks deadlines are defined by the "**slack**" parameter

$$\text{Deadline} = \text{Release-date} + \text{Average task duration} \times \textbf{Slack}$$

On existing BOINC
projects the average slack is ≈ 60
Third proposal:

Dynamic slack

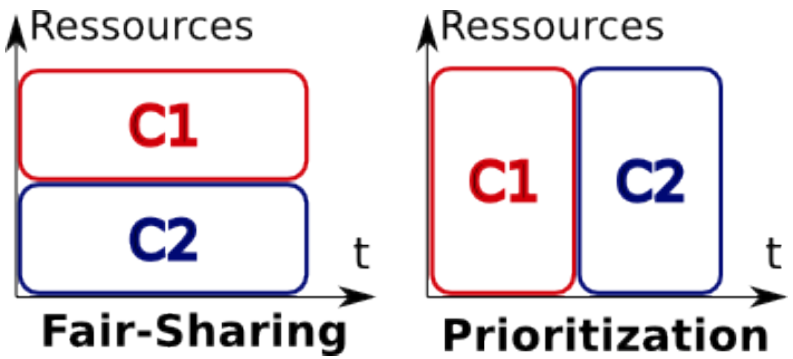
Adapt
the slack value in order to
influence client reactivity



Throughput & Response-time Optimization

World Community Grid: the prototype of an umbrella project
Scheduling is done by instantaneous *Fair-Sharing* among applications

$\longrightarrow$ Known to be good for throughput, not for response-time



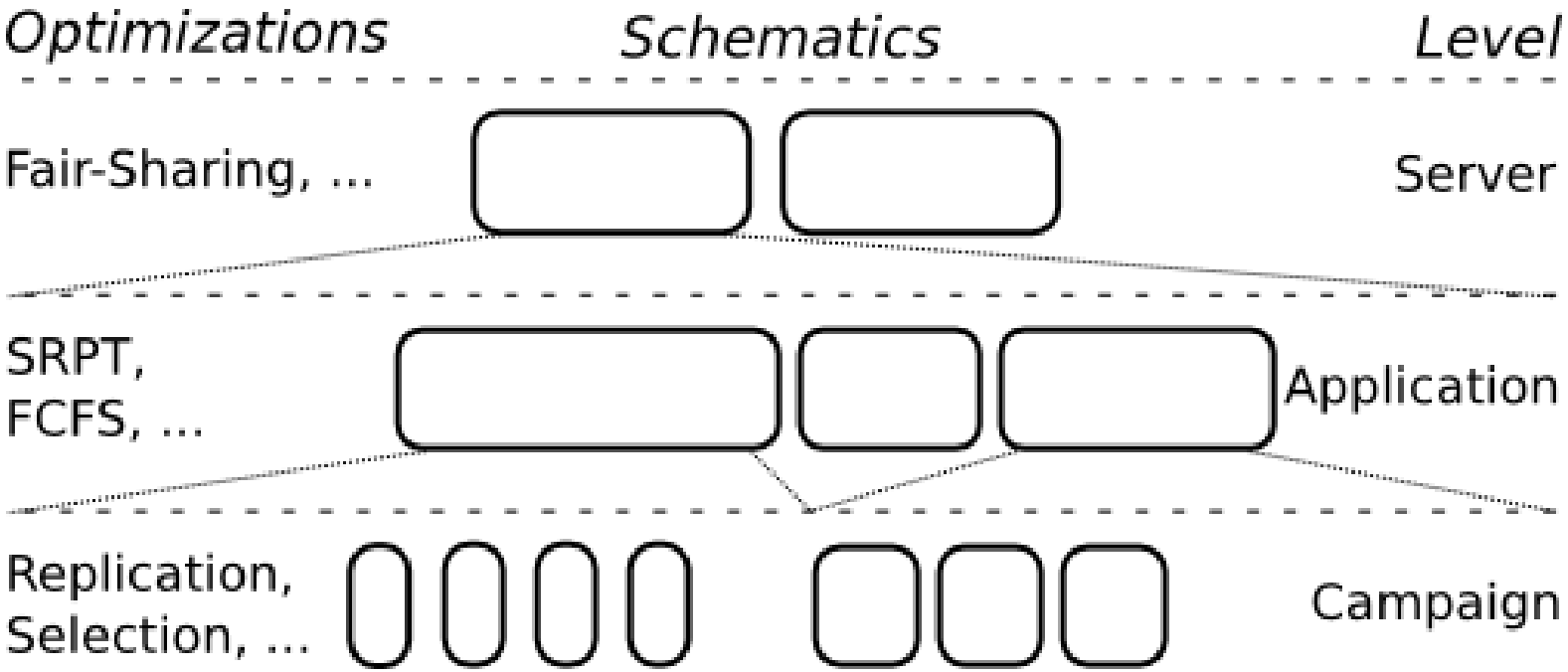
Other strategies are:

- *First-Come First-Serve* optimal for maximum response-time
- *Shortest Remaining Processing Time* optimal for sum of response-times

Section outline

1. Introduction & Research Context
2. Volunteer Computing in a Nutshell
3. Server-side scheduling architecture
   - A Hierarchical Point-of-View
   - Pressurized Scheduling
4. Evaluation and Simulation
5. Conclusion

Campaigns, Applications and a Server



Pressure Measure

We can make three observations:

1. Being fair can be seen as equally sharing the resources over time
2. Between release-date and finish we ought to progress continuously on the number of completed tasks to keep users satisfied
3. In the case we specify a deadline for a campaign all time spent computing after this moment should be avoided

- Late on schedule $\longrightarrow$ high pressure
- Early on schedule $\longrightarrow$ low pressure

Section outline

1. Introduction & Research Context
2. Volunteer Computing in a Nutshell
3. Server-side scheduling architecture
4. Evaluation and Simulation
   - Toolkit
   - Experimental Results
5. Conclusion

SimGrid

Actual modeling of the BOINC server-client behavior is extremely complex

$\rightarrow$ We resort to realistic simulation with the SimGrid toolkit

SimGrid has already been thoroughly tested and used within the distributed computing community

One of the strengths of SimGrid is its scalability allowing to easily simulate systems of a 100,000 nodes and the fact that an BOINC simulator already existed in SimGrid

The BOINC simulator

Was directly inspired by the BOINC source code
Its has been validated by traces from existing BOINC projects

Features of the old simulator:

- No / Very low configurability, only *Fair-Sharing* policy automatically used

Features of the new simulator:

- Allows to simulate:
  - 3 application policies (*Round-Robin*, *Fair-Sharing*, *Pressurized*)
  - 5 campaign policies (*RR*, *FS*, *Pressurized*, *FCFS*, *SRPT*)
  - Mixing of policies on seperated applications
- Implements client selection, aggressive replication and dynamic slack (possibility to activate each one separatly)

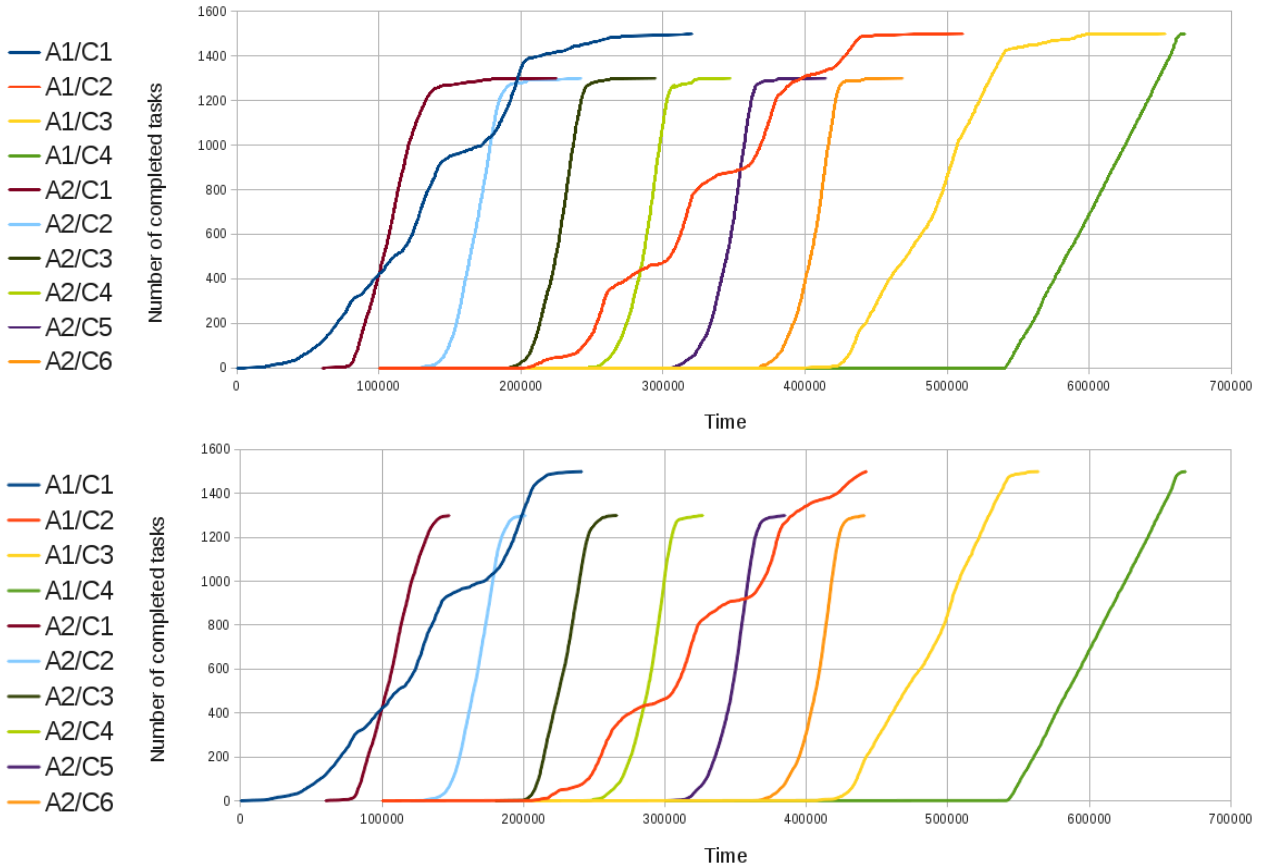
Experimental configuration: A Toy Example

For our experiments we used the following setup:

- 100 computational hosts with discontinuous availability (FTA)
- 2 applications each with multiple identical campaigns
- Applications interfere as do the campaigns of the same application

Application	First release	Release interval	Tasks per campaign	Quorum	Average task duration
1	0	100000	1500	3	≈2200
2	60000	60000	1300	2	≈1500

Effect of Campaign-level Optimization



FS/FCFS scheduling without campaign-level optimization or with client selection and dynamic slack

Comparison of Scheduling Policies

We can draw a multi-user performance chart based on our toy example

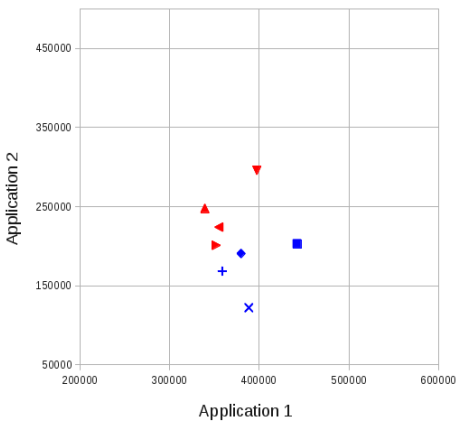


Figure: No optimization for tasks

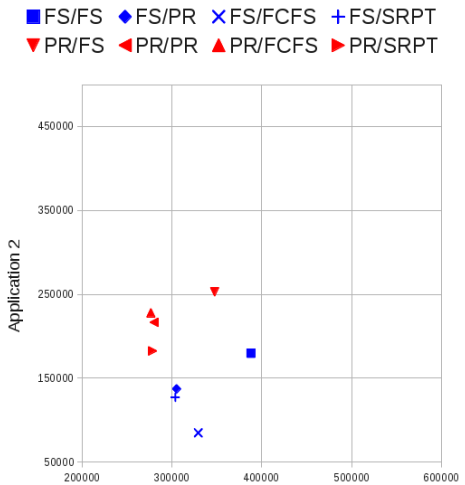


Figure: Client selection and dynamic slack

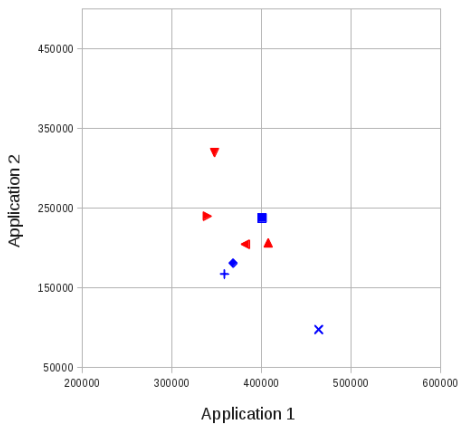


Figure: All optimizations at task-level

Average response-time for each application/campaign policy combination and campaign-level optimization

Comparison of Scheduling Policies, ctd.

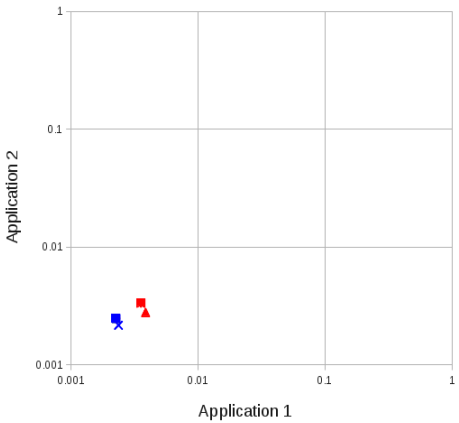


Figure: No optimization for tasks

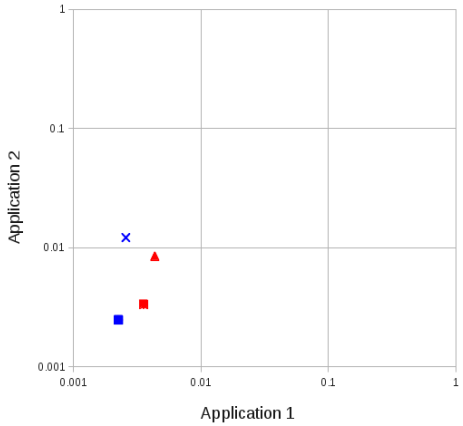


Figure: Client selection and dynamic slack

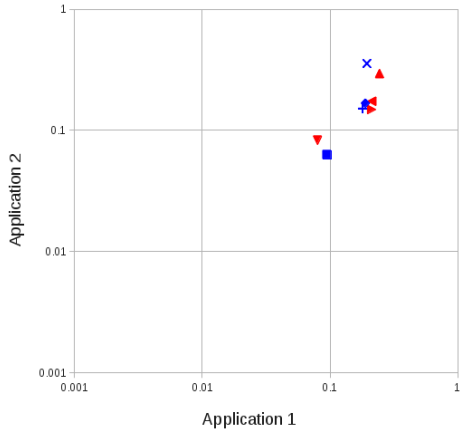


Figure: All optimizations at task-level

Waste rate for each application/campaign policy combination and campaign-level optimization

Conclusion

State-of-the-art:

- Analysis of the evolutions in BOINC usage towards interactivity and multi-user / multi-campaign use-cases
- Review of the existing BOINC software and its usage in the World Community Grid
- Summary of some general and simple scheduling policies and results

Our contributions:

- A new campaign-level optimization that seems to lower response-time
- A hierarchical structure for BOINC "umbrella" projects
- A new scheduling measure: Pressurized scheduling
- Fully configurable simulator code for further simulations