Improving capabilities of P2P Volunteer Computing Platform

Maximiliano Geier[1] and Esteban Mocskos[1,2]

[1]Departamento de Computación
Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires

C S C

[2]Consejo Nacional de Investigaciones Científicas y Técnicas

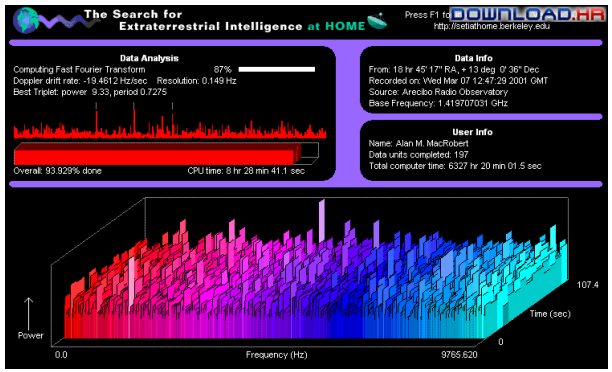
November 7th, 2013

Introduction

- The infrastructure available for e-science involves a wide range of sites and architectures
- Volunteer Computing consists of donating computational resources in order to create a grid that can solve complex problems
  - BOINC
  - OurGrid
- Bag-of-Tasks is an application model that involves arranging several independent uniprocessor tasks that require no communication with each other
- BoT is especially well suited for heterogeneous environments such as Volunteer Computing platforms

BOINC

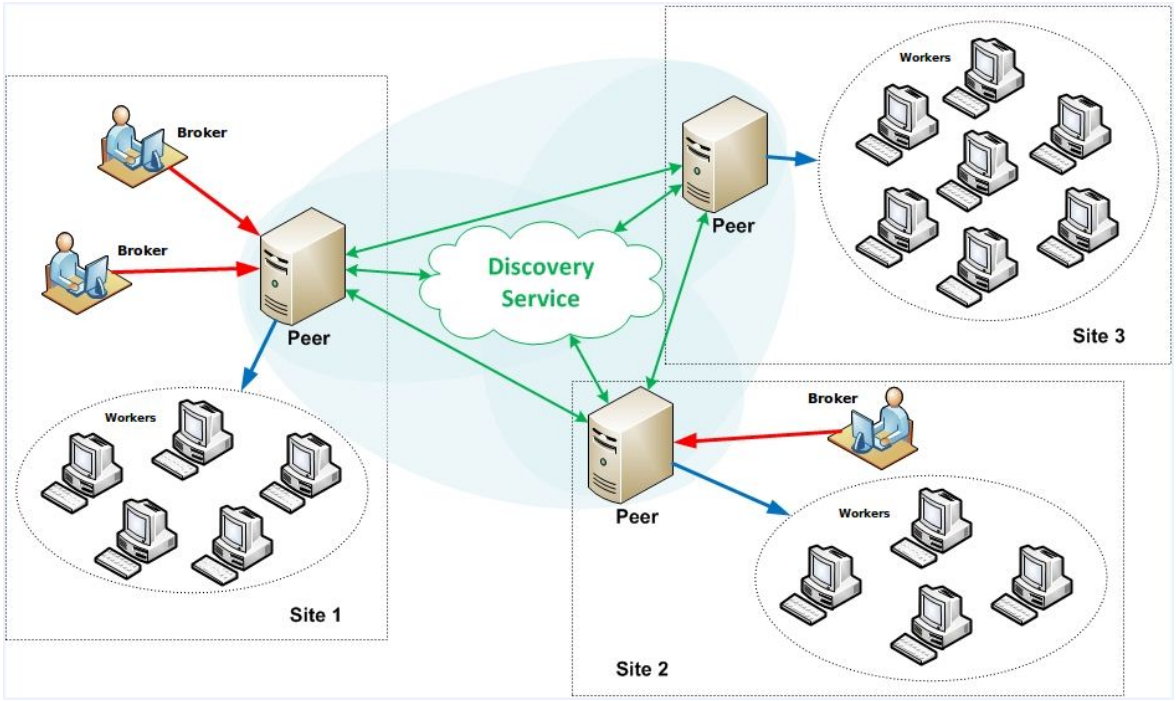
- Volunteer Computing Platform
- Developed at UC Berkeley, California
- 237,241 volunteers, 486,116 computers @ 8.296 petaFLOPS (November 2013)
- Volunteers donate their CPU power to a specific project
  - SETI@Home
  - LHC@Home
  - Docking@Home



OurGrid

- Volunteer Computing Platform
- Developed at UFCG, Campina Grande, PB, Brazil
- More than 50 papers based on its results
- Starting collaboration between UFCG and UBA
- Volunteers donate their CPU power to run computing tasks

OurGrid architecture



- Written in Java
- Four components: Broker, Worker, Peer, Discovery Service
- Communication implemented using XMPP
- Broker: submits and monitors jobs
- Worker: runs assigned tasks
- Peer: interacts with the on-site discovery service

Problem description

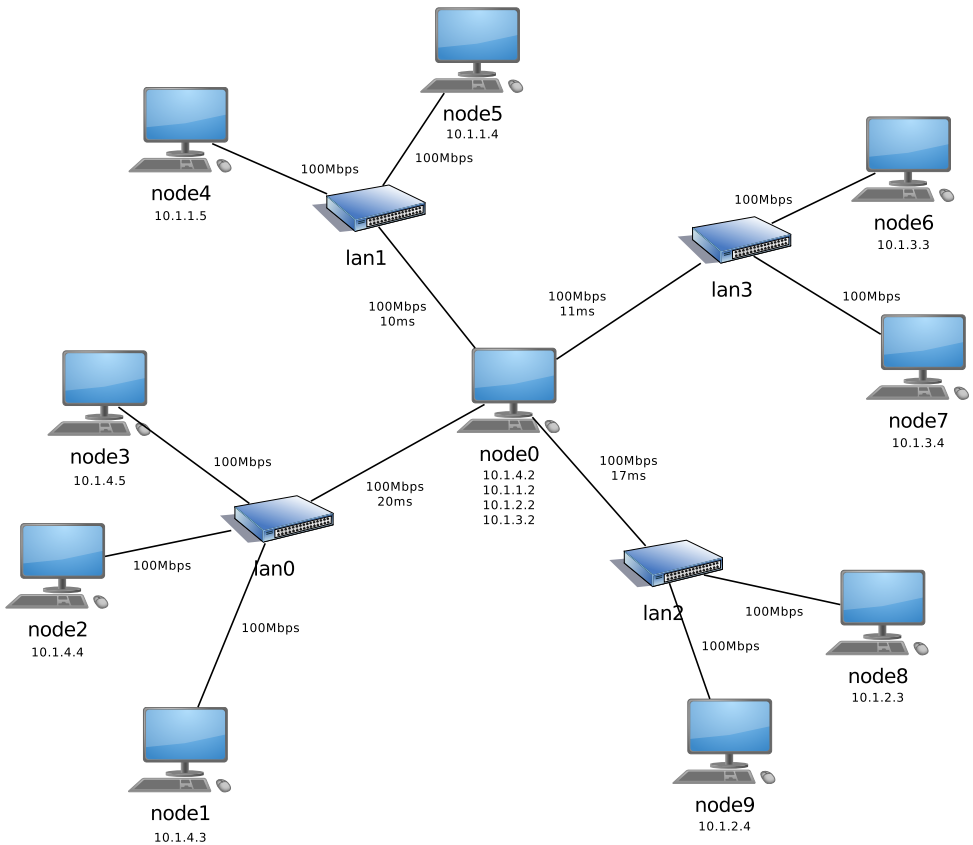
- OurGrid can only manage Bag-of-Tasks
- We want to increase the scope of the platform by allowing it to run tasks with weak dependencies (i.e. tasks that need to communicate but only seldom)
- This new scenario could be enabled by clustering nodes by latency and assigning task dependencies in the same cluster
- OurGrid schedules tasks using a replication schema on a first-discovered-resource basis, which we assume to be random with respect to node latency

Methodology

- Simple synthetic MPI-like application runs on a subset of nodes
- Each scenario consists of a network model, a node selection policy and simulation parameters for the MPI application
- The network model consists of interconnected nodes and switches, which include latency information on each link
- Two different network sizes
  - 10 nodes – 4 interconnected LANs
  - 33 nodes – 10 interconnected LANs

Network model

- Each network is modeled as a graph
- Nodes are either hosts or switches
- Edges are network links, their weights represent the theoretical latency
- Path latency is calculated using Dijkstra's shortest path algorithm

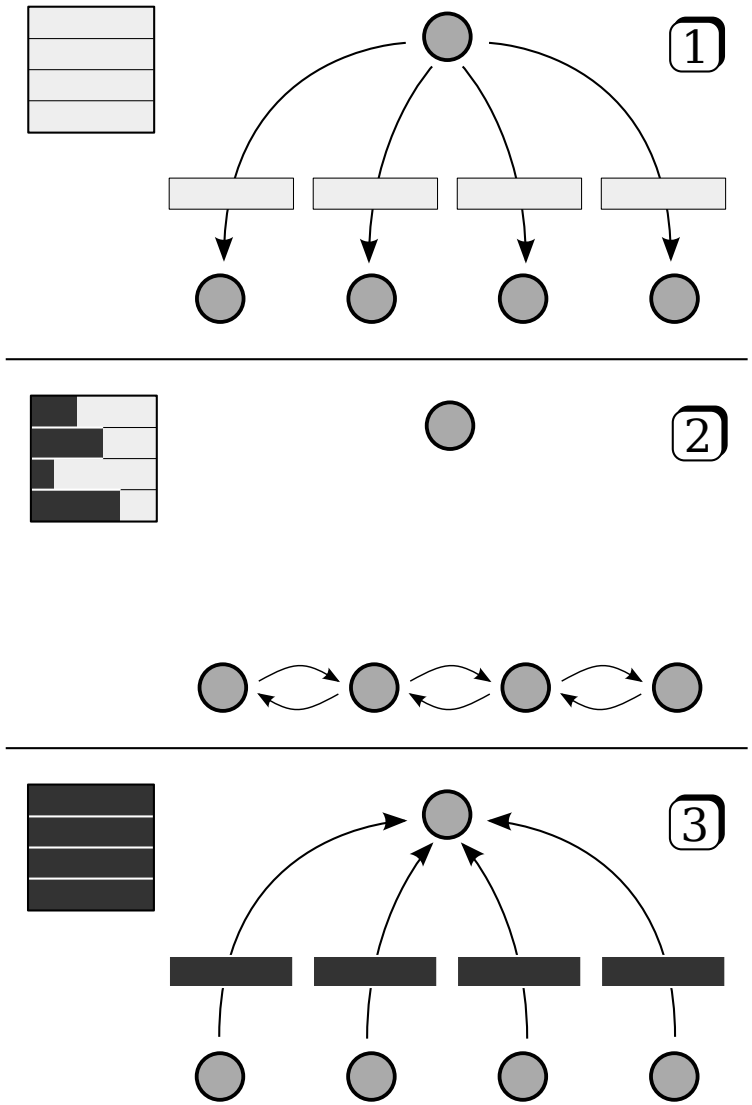


Node selection policies

- Random
  - `random`: randomly selected permutations of nodes
- Neighbour distance
  - `neighbour`: minimization of maximal pairwise distance between nodes
- Master proximity
  - `master`: minimization of average distance to the first node (*master* node)
  - `master*`: minimization of maximal distance to the first node
- Combined policy: second criteria is used to decide among subsets with the same value for the first one
  - `best`: combined `neighbour` and `master`
  - `best*`: combined `neighbour` and `master*`

Application model

1. **Setup**: master assigns data to worker nodes
2. **Space iteration**: computes data, exchanging frontier information with neighbours
3. **Time iteration**: worker nodes send data back to master, which evaluates a convergence criteria

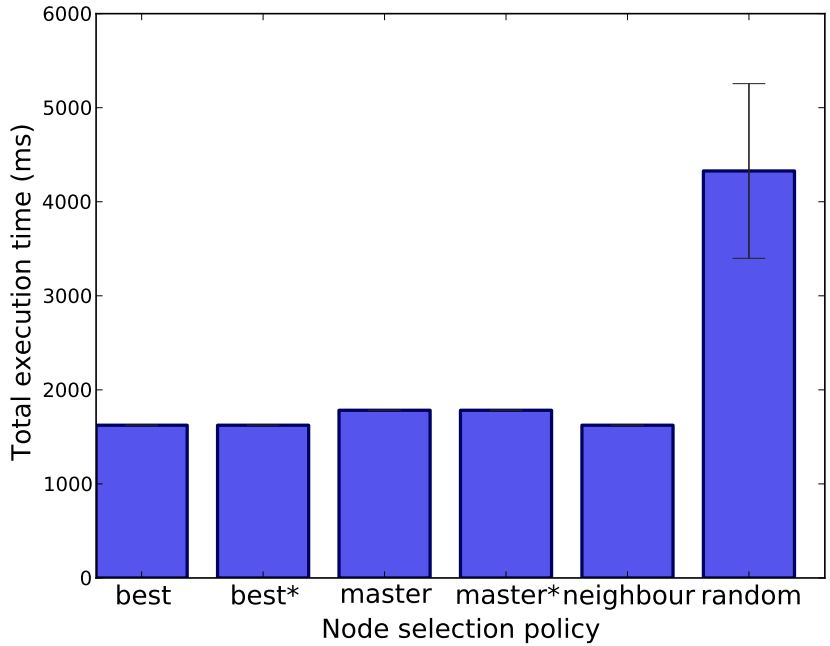


Data generation

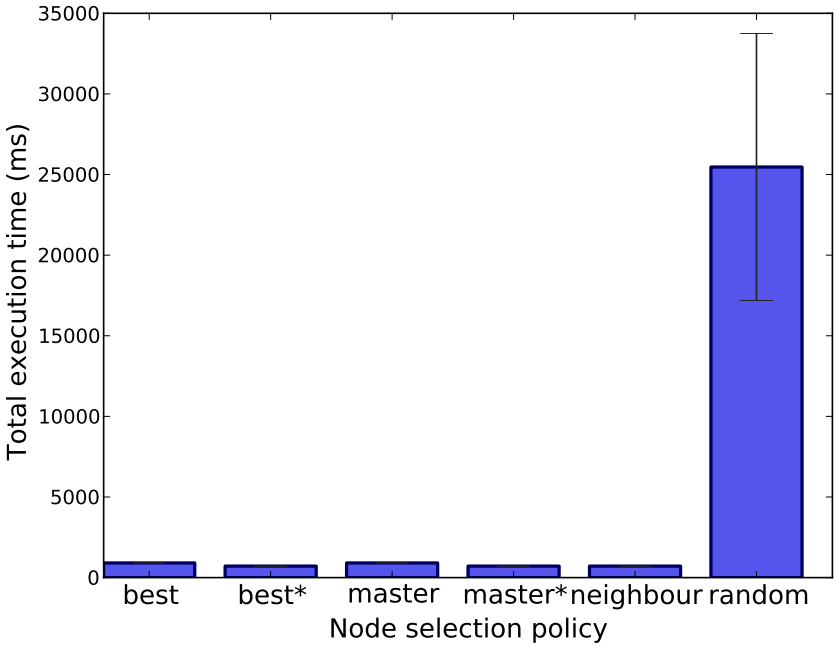
- Subsets of nodes of size 5 corresponding to each selection policy were calculated for each network
- Each subset was fed to the simulation to evaluate the total execution time based on the theoretical latency of each path and the application model
- Application model parameters: 5 space iterations, 5 time iterations
- For random, as many different subsets as the square of the total number of nodes were evaluated, producing average and deviation data

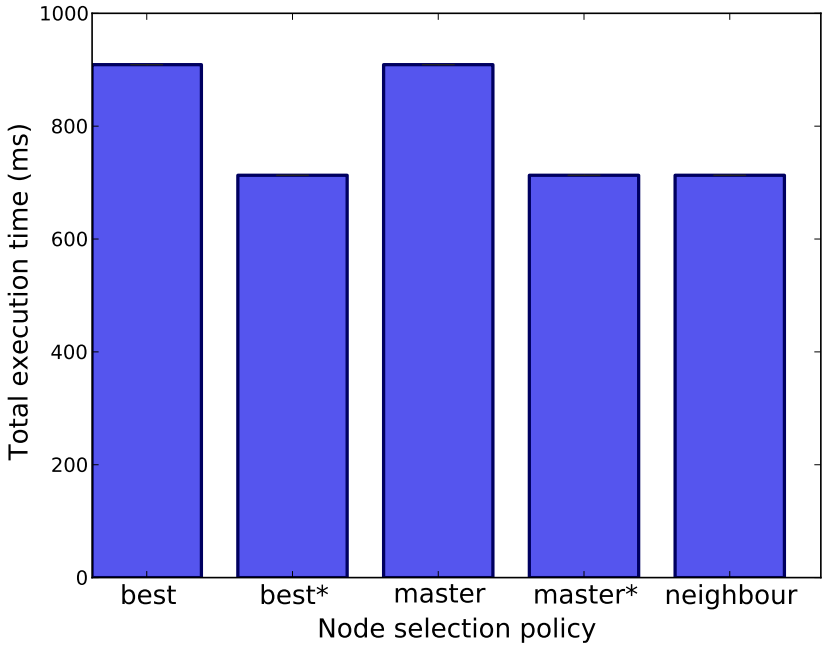
Results

- On the 10-node network, random shows great deviation, but the analyzed selection policies improve on it at least two-fold
- *Neighbour distance* shows the best results



Results (cont.)





- On the 33-node network, the differences between random and the rest of the selection policies are even greater, due to the fact that the average path length in this network is much larger
- Again, *neighbour distance* shows the best results

Conclusions

- We have shown that it is possible to schedule tasks with weak dependencies in a way that drastically decreases the execution time in an heterogeneous environment, given that latency information for the underlying network is available
- This information could be used to extend the capabilities of volunteer computing systems such as OurGrid beyond Bag-of-Tasks
- This shows that it is worthwhile to pursue this line of work, widening the application scope, network sizes and other parameters

Thank you for listening!

Questions?