Distributed BEAGLE:
An Environment for Parallel and Distributed Evolutionary Computations

Christian Gagné, Marc Parizeau, and Marc Dubreuil

Département de génie électrique et de génie informatique

Laboratoire de vision et systèmes numériques

UNIVERSITÉ LAVAL

Québec (Québec), Canada

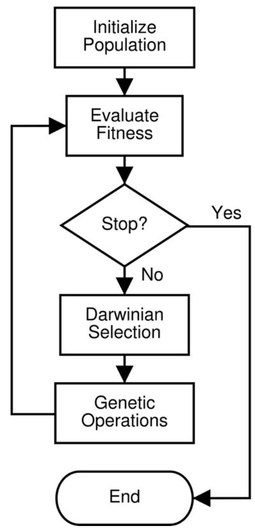
Outline

- Evolutionary Computations (EC)
- Parallel and Distributed EC
- Master-slave architecture
- Deployment scenario
- Proposed implementation

2

Evolutionary Computations (EC)

- Simulation of natural evolution on computers
- Generic problem-solving method
  - Solutions represented by data structures
  - Objective function (fitness)
- Population of solutions that evolve over time
- Optimization, machine learning, automatic design



3

Four Flavors of EC

- Genetic Algorithms (Holland, 1975)
  - Vectors of characters: <10011000111>
  - **Crossover**, mutation, selection
- Genetic Programming (Koza, 1992)
  - Solutions = LISP *s*-expressions (programs)
- Evolution Strategy (Rechenberg, 1973)
  - Vectors of floating-point numbers
  - Mutation strategy
- Evolutionary Programming (Fogel et al., 1966)
  - At first finite state machines, later vectors of floats
  - Mutation specific to the representation

4

Implementing EC

Data structures

- Population of solutions
  - Bit strings (GA)
  - Graph representing programs (GP)
- Containers and dynamic polymorphism

Algorithms

- Evolutionary loop with operators
  - Fitness evaluation
  - Genetic operations
- *Strategy* design pattern

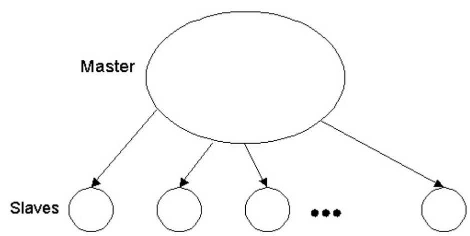
5

Parallel and Distributed EC = PDEC

- EC need huge CPU resources
- EC are implicitly parallel: a population of independent solutions evolving in parallel
- For real world problems, solution fitness evaluation is the computation bottleneck
- PDEC is a hot topic: Beowulf clusters are cheap and well adapted for PDEC

6

Master-Slave



- Master stores the whole population and applies genetic operators
- Master distributes individuals to the slaves for fitness evaluation

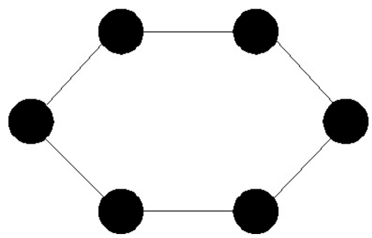
7

Pros and Cons of Master-Slave

- Pros
  - Simple transposition of sequential model
  - Node can be added/removed dynamically
  - Robust to slave failures
  - Simplifies data collection/analysis
- Cons
  - If the master crashes the whole system goes down
  - Communication overhead
  - May not scale well when the master is overloaded
  - Synchronization overhead for lagging slaves

8

Island-Model



- Isolated evolutions with a migration process
- Encourages diversity and prevents premature convergence
- 1 CPU = 1 population

9

Pros and Cons of Island-Model

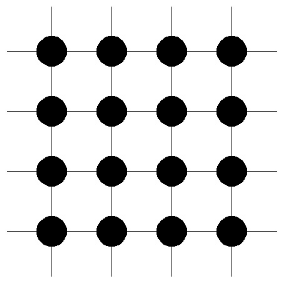
- Pros
  - Scales very well
  - Low communication overhead
  - Robust to failures (willing to lose small populations)
  - Higher diversity: isolated populations with migration
- Cons
  - Load balancing on heterogeneous networks
  - Dynamic reconfiguration of network
  - Evolution cannot be reproduced
  - Difficult data collection/analysis

10

Fine Grained & Hierarchical Hybrid

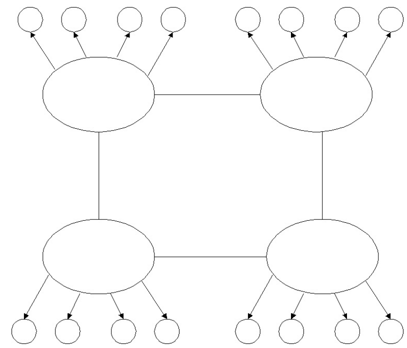
Fine Grained

- Populations spatially distributed on processors
- One individual per processor (SIMD)



Hierarchical Hybrid

- Hybrid of master-slave and island-model



11

Designing a PDEC System

- Networks of computers
  - Beowulf clusters
  - LAN of heterogeneous workstations used during idle time (screen-saver)
- Processing nodes dynamically added/removed
  - Hard failures: system crash/reboot, network problem
  - Soft failures: user deactivates the screen-saver

12

Options

- Master-slave
  - Communication bottleneck
  - Robust to failures: task of a slave can be easily redispatched
- Island-model
  - Scales very well, peer-to-peer, WAN
  - Independent populations (1 proc. = 1 pop.)
  - MTBF << evolution time?

13

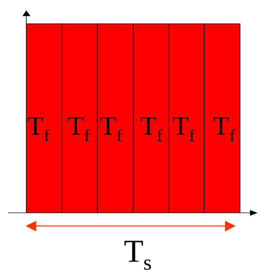
Outline

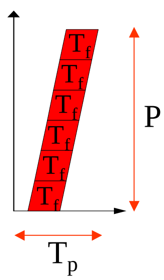
- Evolutionary Computations (EC)
- Parallel and Distributed EC
- Master-slave architecture
- Deployment scenario
- Proposed implementation

14

Speedup of Master-Slave

$$speedup = \frac{T_s}{T_p} \qquad T_s = NT_f$$





15

Parameters

- N: population size
- P: number of processors (slaves)
- T_f: average fitness evaluation time
- T_c: average communication time
- T_l: average connection latency
- S: average number of solutions composing a distribution set
- C: number of evaluation cycle
- K: number of failures observed during a generation

16

Distribution Policies

- S = number of solutions sent to each slave during each communication cycle
- Two common policies:
  - P processors, P sets of size N / P (S = N / P)
  - one-by-one (S = 1)
- Third option: adaptive S

17

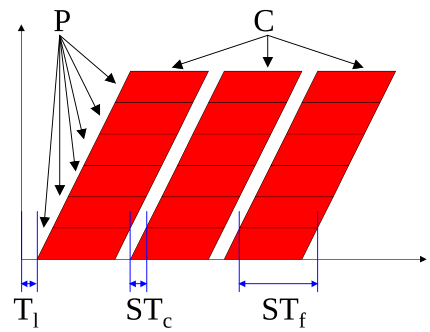
Assumptions

- Computers with similar performance (variance of S is small)
- Averaged time values
- Constant number of processors

18

Illustrating Values

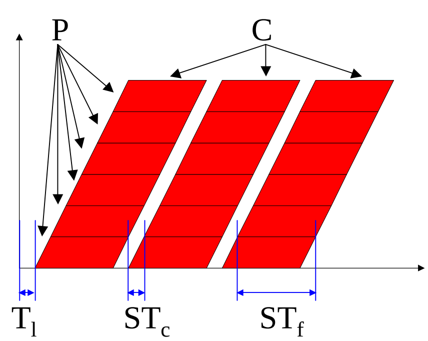
- S: size of sets
- P: # of processors
- C: # of evaluation cycles
- T_f: fitness time
- T_c: transmission time
- T_l: latency time



19

Mathematical Modelization

$$T_p = \underbrace{CST_f}_{\text{computation}} + \underbrace{CPST_c}_{\text{communication}} + \underbrace{CT_l}_{\text{latency}} + \underbrace{T_k}_{\text{failures}}$$



20

Failure Delay

$$T_k = \begin{cases} 0 & K = 0 \\ \underbrace{(1-0.5^K)ST_f}_{\text{synchronization}} + \underbrace{KST_c}_{\text{comm.}} + \underbrace{T_l}_{\text{latency}} & K \in [1, P] \end{cases}$$

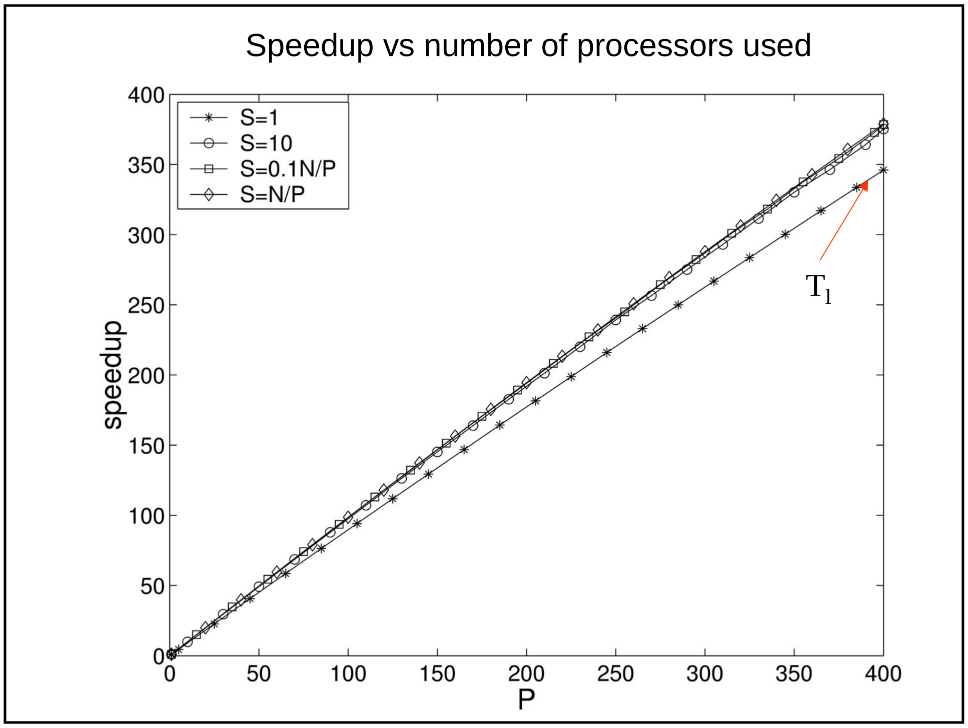
- K: the number of observed failures
- Synchronization term: under the assumption that failures are independent, follow a Poisson process, and happen half-way through the fitness evaluation process

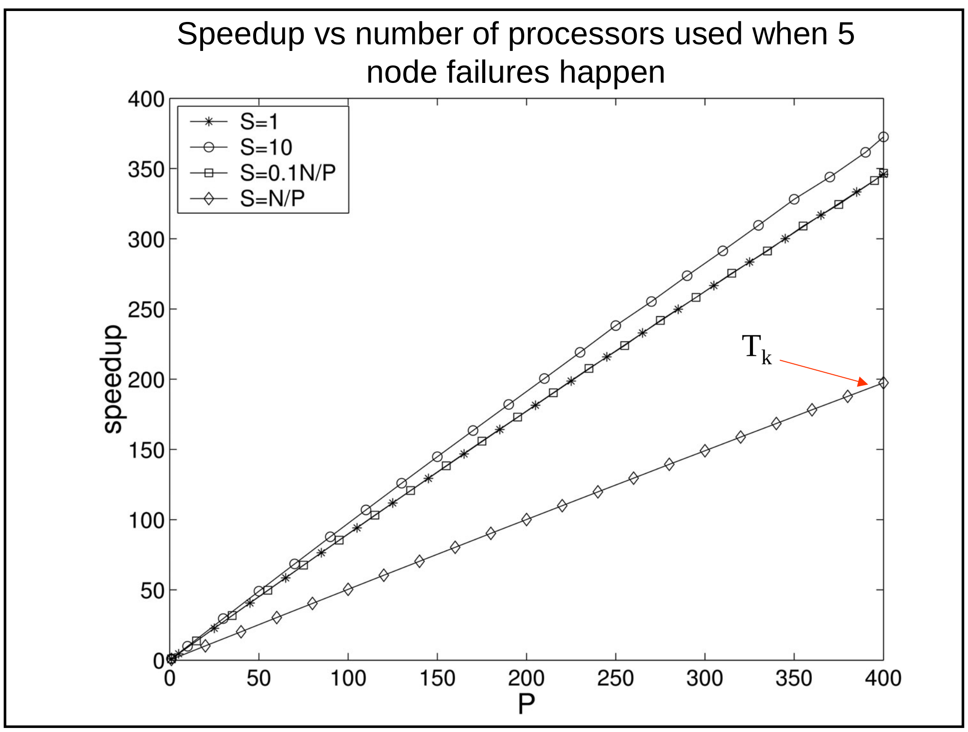
21

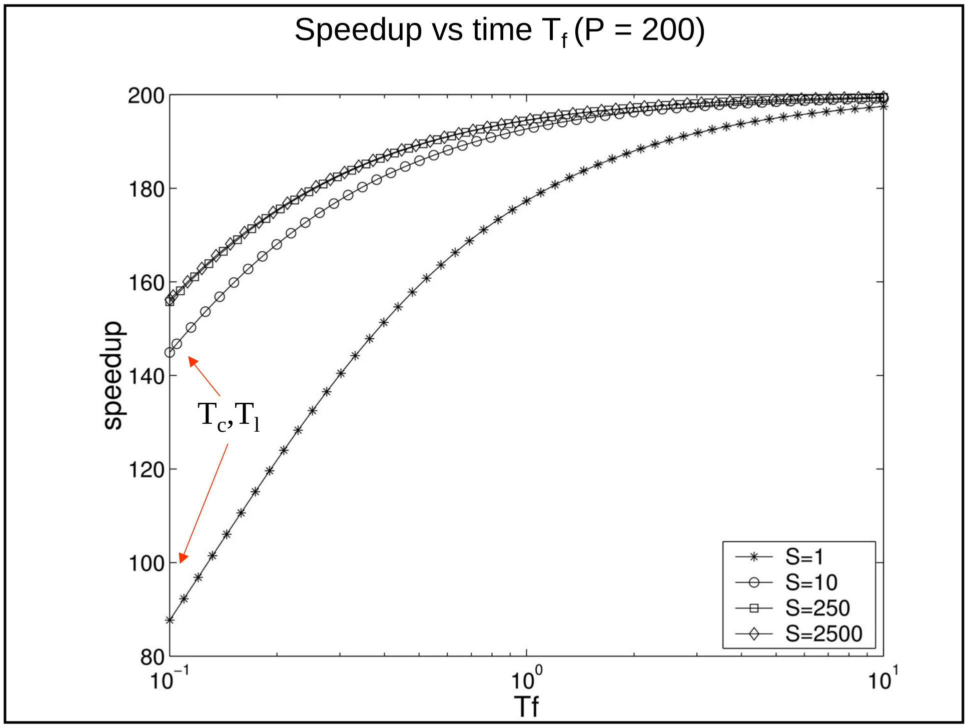
Plausible Scenario: Beowulf

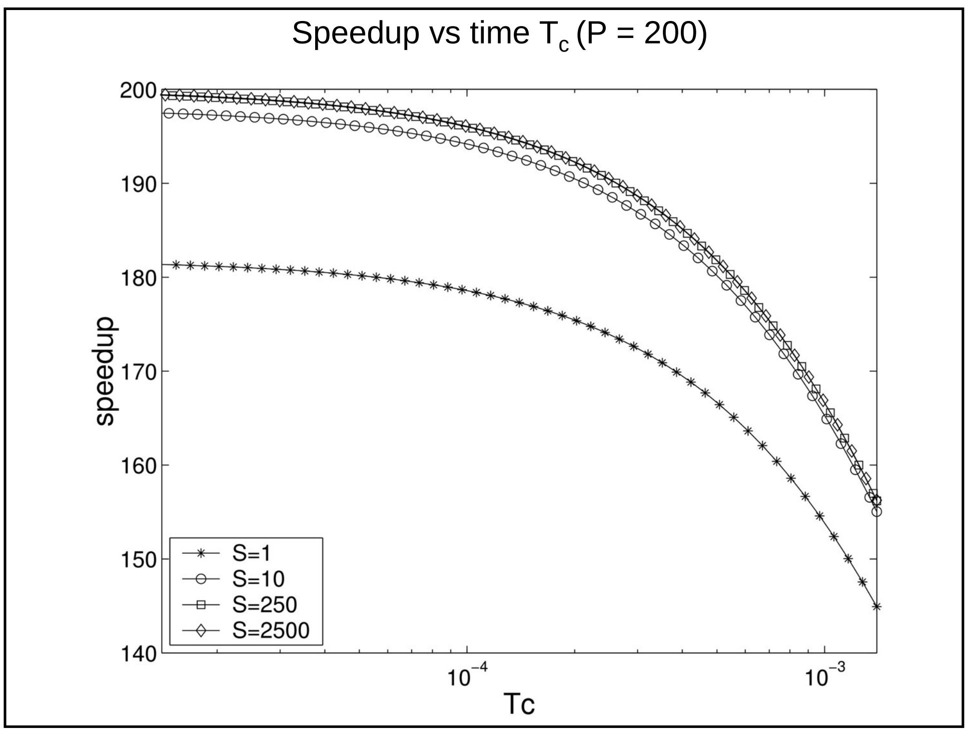
- 100 Base-T switches (7MBps effective bandwidth)
- Average fitness evaluation time T_f = 1 s
- Solution = 1KByte -> T_c = 0.14 ms
- Average connection latency T_l = 0.1 s
- 500 000 solutions
- Between 1 and 400 processors
- Size of sets S = {1, 10, 0.1N/P, N/P}

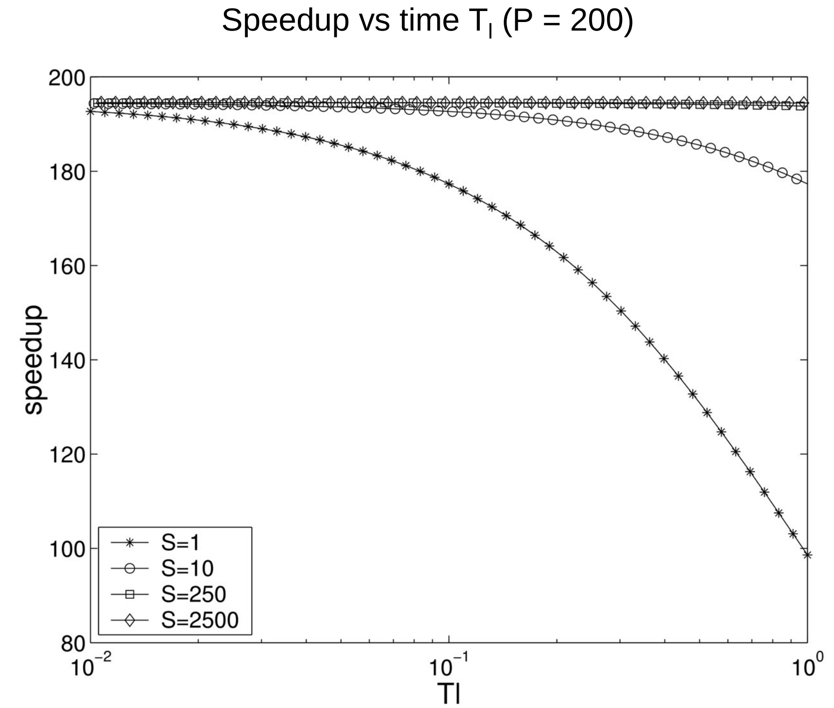
22


Speedup vs number of processors used
speedup
P
S=1
S=10
S=0.1N/P
S=N/P
T_l


Speedup vs number of processors used when 5 node failures happen
speedup
P
S=1
S=10
S=0.1N/P
S=N/P
T_k


Speedup vs time T_f (P = 200)
speedup
Tf
T_c, T_l
S=1
S=10
S=250
S=2500
200
180
160
140
120
100
80
10^-1
10^0
10^1


Speedup vs time T_c (P = 200)
speedup
Tc
S=1
S=10
S=250
S=2500
200
190
180
170
160
150
140
10^-4
10^-3



Communication Bottleneck

- In this scenario, master-slave scales to more than 7000 processors before network saturation (speedup around 3500)
- Use of intermediary size sets S necessary to achieve best performances (trade-off between latency and failures penalty)

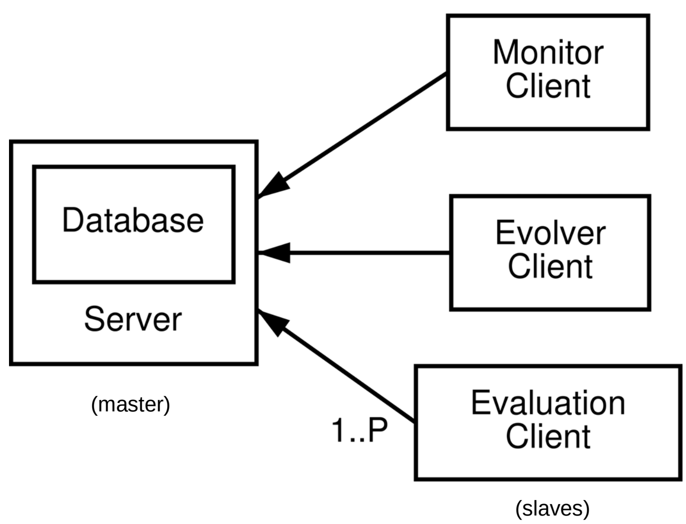
28

Outline

- Evolutionary Computations (EC)
- Parallel and Distributed EC
- Master-slave architecture
- Deployment scenario
- Proposed implementation

29

Distributed BEAGLE



30

Characteristics

- Dynamic adjustment of the size of sets S based on previous results
- Redistribution of data when slaves are lagging
- Support for multiple populations: island-model with synchronous migration can be simulated to promote diversity
- Independent of the EC system and algorithm used

31

Technologies

- Coded in C++
- SQL database for data persistency
- Communication based on TCP sockets
- Messages exchanged between the clients and the server encoded in XML

32

State of Developments

- There is already a working prototype
- Public release as open source project
- Integrated with the C++ EC framework Open BEAGLE (http://www.gel.ulaval.ca/~beagle)

33

Conclusion

- Master-slave is usable for LAN of workstations with limited availability
- Master-slave scales well (up to a certain point)
- Size of set S should be dynamically adjusted
- Distributed BEAGLE: a master-slave architecture for networks of computers with limited availability

34