Building a Distributed Genetic Algorithm with the Jini Network Technology

Brian Zorman
(Gregory M. Kapfhammer and Robert Roos)

Sixth Annual Jini Community Meeting
Boston • June 17-20, 2002
Problem Analysis

- Genetic Algorithms:
  - **Pros:** robust and efficient
  - **Cons:** execution cost and Quality of Solution (QoS)

- Possible solution: how can we harness the benefits of distributed computing frameworks?

- Can we reduce cost of execution and improve quality of solution with a distributed genetic algorithm (DGA)?
Bridging the Gap: Distributed Genetic Algorithms

Genetic Algorithms:
1.) Execution cost
2.) Lack of diversity

Distributed Systems:
1.) Resource Sharing
2.) Concurrency
3.) Scalability
4.) Openness
Exploring Punctuated Equilibrium

• The theory of punctuated equilibrium:
  – An isolated environment can reach a point of stability
  – The injection of new individuals could cause rapid evolution

• Could we design a distributed system to simulate this theory?

• How can the Jini network technology and the JavaSpaces object repository help us to build this distributed system?
Designing the Models

- Examined two popular models: **master-worker** and **island**

- Chose combination of master-worker and island models
  - **Master-worker**: parallel execution and simplicity
  - **Island model (punctuated equilibrium)**: parallel execution and additional diversity
High Level Architecture: Entities in the “Simple” Model

- DistributionSpace
- DiversitySpace

- Initial Machine
- RM1
- RM2
- RM3
- \ldots
- RMn
"Simple" Model: Distribution Phase

DistributionSpace

Initial Machine → RM1 → RM2 → RM3 → ... → RMn

DiversitySpace
"Simple" Model: Pre-migration

Initial Machine

DistributionSpace

DiversitySpace

RM1

RM2

RM3

…

RMn
“Simple” Model: Migration

Initial Machine

DistributionSpace

DiversitySpace

RM1

RM2

RM3

…

RMn
“Simple” Model: Post-convergence

Initial Machine → DistributionSpace

RM1 → RM2 → RM3 → ... → RMn → DiversitySpace
Simple Model Performance Bottleneck

- No explicit synchronization between remote machines
- Potentially, each remote machine could migrate with JavaSpace at the same time!
- In some sense, this causes each worker to “wait in line” in order to perform migration!
- While each worker is waiting there is no computation!
- Designed “Complex” Distributed System Model (CDSM) in an attempt to reduce this bottleneck
High Level Architecture: Entities in the “Complex” Model

Initial Machine

- MM1
- MM2
- ... (omitted)
- MMn

DistributionSpace

- MS1
- MS2
- ... (omitted)
- MSn

- RM1
- RM2
- ... (omitted)
- RMn
“Complex” Model: Distribution Phase

Initial Machine → DistributionSpace

MM1 → MS1 → RM1

MM2 → MS2

... → MSn

MMn → RMn
“Complex” Model: Pre-migration

Initial Machine

DistributionSpace

MM1 → MS1
MM2 → MS2
... → ... → MSn

RM1
RM2
... → RMn
“Complex” Model: First Migration Phase

Initial Machine

- MM1
- MM2
- ..
- MMn

DistributionSpace

- MS1
- MS2
- ..
- MSn

- RM1
- RM2
- ..
- RMn
“Complex” Model: Subsequent Migration Phases

Initial Machine

- MM1
- MM2
- ...
- MMn

DistributionSpace

- MS1
- MS2
- ...
- MSn

- RM1
- RM2
- ...
- RMn
"Complex" Model: Post-convergence

Initial Machine

DistributionSpace

MM1

MS1

RM1

MM2

MS2

RM2

.....

MMn

MSn

RMn
“Complex” Model Observations

- Maintains the functionality of the “Simple” model

- Requires dedicated MigrationMachines and MigrationSpaces

- Explicit synchronization mechanism used so that chances of more than one remote machine migrating with the same JavaSpace at the same time is greatly reduced

- Multiple MigrationSpaces minimally reduce the overall diversity that any given remote machine has access to; however, this cost is small when compared to other gains!
Experimental Framework

- **Goal**: analyze the design and performance of the two models, and then compare the best version to sequential GA

- Selected open source GA written in Java that “solves” the Knapsack Problem
  - Knapsack problem is provably NP-complete

- **Knapsack Problem Statement**: Given a set of weights and knapsack capacity: find best combination of weights that fit inside the knapsack
Testbench Description

- 8 testsets of increasing levels of difficulty

- **Range of weight values:**
  0 – 5000

- **Number of weights:**
  500 – 1200

- **Number of machines**
  - SDSM: {2,4,6,8}
    - Requires RemoteMachines
  - CDSM: {2,4,6,8}
    - Requires RemoteMachines, MigrationMachines, MigrationSpaces

- **GA parameters:**
  - **Termination condition:** best solution remains constant after 75 generations
  - **Crossover:** at every generation
  - **Mutation:** at every generation
  - **Migration:** 30% of population every 30 generations, starting at generation 60
Measurements and General Observations

- **Execution time:** The CDSM reduces the execution time of the DGA when compared to the SDSM. Generally, overall execution time increases as we add machines to the CDSM.

- **Computation–to–Communication ratio:** CDSM increases this ratio when compared to the SDSM. The addition of machines to the CDSM reduces this ratio.

- **Diversity:** The potential for a higher quality solution increases as we move from the SGA to the CDSM and then as we add more machines to the CDSM.

- **Quality of Solution:** The QoS for the CDSM is always higher than the SGA. Generally, the QoS is higher in the CDSM as we add machines.

- **Generations–per–Second:** The CDSM can compute more Gen/Sec than the SDSM. Generally, adding more machines to the CDSM increases the Gen/Sec.
SDSM vs. CDSM: Execution time
SDSM vs. CDSM: Computation-to-Communication Ratio
SDSM vs. CDSM: Generations/Second
CDSM vs. SGA: Quality of Solution
CDSM vs. SGA: Execution Time

The graph compares the execution time of CDSM and SGA with varying numbers of machines (2, 4, 6, 8) for different tasks (labeled from 1 to 8). The Y-axis represents execution time in milliseconds, while the X-axis represents the tasks. The bars indicate the execution time for each task under different machine configurations.
CDSM vs. SGA: Computation-to-Communication
CDSM vs. SGA: Population Diversity

![Graph showing population diversity comparison between CDSM and SGA across different machine counts (2, 4, 6, and 8)].

Legend:
- SGA
- 2 mach.
- 4 mach.
- 6 mach.
- 8 mach.
CDSM vs. SGA: Generations-per-Second
Future Possibilities: Distributed GA Framework

- Potential advantages of a DGA framework:
  - Could be integrated into existing Java GA frameworks
  - Java provides GA portability across operating systems
  - Jini and JavaSpaces offer openness, scalability, fault tolerance
  - GA developers could easily distribute their GA just to “see what happens”

- DGA framework would require an approach for automatically and transparently starting and terminating remote workers

- Various users should be able to donate their resources; our DGA can make use of “idle time” on various university machines

- Potentially, we could develop simple applet for visibility and learning
Concluding Remarks

• Investigated feasibility of using Jini and JavaSpaces to build a distributed genetic algorithm

• Proposed, implemented, and empirically evaluated a simple and a complex distributed system model (SDSM and CDSM)

• SDSM bottleneck was a serious concern that prompted the investigation of a new model that removed JavaSpaces interaction bottlenecks

• CDSM outperformed SGA in quality of solution, diversity, and generations per second

• SGA only outperformed CDSM in execution time (mostly due to early convergence)