ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Concept Location with Genetic Algorithms: A Comparison of Four Distributed Architectures

Fatemeh Asadi
Giuliano Antoniol
Yann-Gaël Guéhéneuc

SOCCER LAB
SOftware Cost-effective Change and Evolution Research Lab

Ptidej
Pattern Trace Identification, Detection, and Enhancement in Java

Content

- Motivations
- Literature Review
- Background
  - Getting Traces, Trace Pruning and Compression
  - Textual Analysis of Method Source Code
  - Applying Search-based Optimization Technique
- Parallelizing GAs
  - Rationale of the Architectures
  - Simple Client Server Architecture
  - Database Configuration
  - Hash-database Configuration
  - Hash Configuration
- Summery of Results
- Conclusion and Future Work

Motivations

- Genetic algorithms (GAs) are attractive to solve many search-based software engineering problems
- GAs are effective in finding approximate solutions when
  - The search space is large or complex
  - No mathematical analysis or traditional methods are available
  - The problem to be solved is NP-complete or NP-hard

- GAs allow the parallelization of computations
  - The evaluation of the fitness function is often performed on each individual in isolation
  - Parallelizing the computations improves scalability and reduces computation time

Motivations

- In this paper, we report our experience in distributing the computation of a fitness function to parallelize a GA to solve the concept location problem.

- We developed, tested, and compared four different configuration of distributed architectures to resolve the scalability issues of our concept location approach
  - A client (master) performs GA operations and distributes the individuals among servers
  - Servers (slaves) perform the computations of the fitness function

Literature Review

- Mitchal et al. used a distributed hill-climbing to re-modularize large systems by grouping together related components by means of clustering techniques
- Mahdavi et al. used a distributed hill-climbing for module clustering

- Parallel GAs are classified into three categories (Stender et al.)
  - Global parallelization: only the evaluation of the individuals' fitness is parallelized (our approach)
  - Coarse-grained parallelization (island model): a computer divides a population into subpopulations and assigns each to another computer. A GA is executed on each sub-population
  - Fine-grained parallelization: each individual is assigned to a computer and all the GA operations are performed in parallel

Background

- Our proposed concept location approach consists of the following steps:
  - Step I – System instrumentation
  - Step II – Execution trace collection
  - Step III – Trace pruning and compression
  - Step IV – Textual analysis of method source code
  - Step V – Search-based concept location

Step I and Step II – Getting the Traces

- Step I – System instrumentation
  - It is performed to generate the trace of method invocations
    - We used our instrumentor, MoDeC
- Step II – Execution trace collection
  - We exercise the system following scenarios taken from user manuals or use-case descriptions

Step III – Trace Pruning and Compression

- Trace pruning
  - Too frequent methods occurring in many scenarios are not related to any particular concept (they are often utility methods)
    - We remove methods having a frequency **Q3 + 2 × IQR** (75% percentile + 2 × the inter-quartile range)

- Trace compression
  - We "collapse" repetitions in execution traces to reduce the search space
  - Examples:

    m1(); m1(); m1(); => m1();

    m1(); m2(); m1(); m2(); => m1(); m2();
  - We perform the collapsing using the Run Length Encoding (RLE)
  - We apply the RLE for sub-sequences having an arbitrary length

Step IV- Textual Analysis of Method Source Code

- The data provided in this part are used to calculate conceptual cohesion and coupling as described by Marcus et al. in 2008 and Poshyvanyk et al. in 2006.

- Our assumptions are

  1) Methods helping to implement a concept are likely to share some linguistic information (see Poshyvanyk et al., 2006)

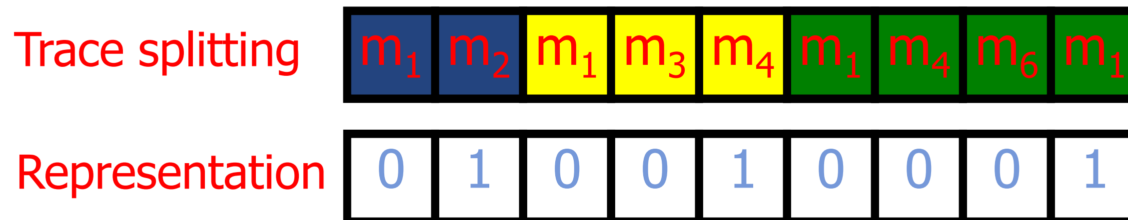
  2) Methods responsible to implement a concept are likely to be called close to each other in an execution trace.

Step IV – Textual Analysis of Method Source Code

- Steps:
  - Extraction of identifiers and comment words
  - Camel-case splitting of composed identifiers: visitedNode -> visited & node
  - Stop word removal (English + Java keywords)
  - Stemming using the Porter stemmer: Visited -> visit
  - Term-document matrix generation
    - Considering each method is as a document
    - Indexing using tf–idf
  - Reducing the term-document space into a concept-document space using Latent Semantic Indexing (LSI)
    - To cope with synonymy and polysemy

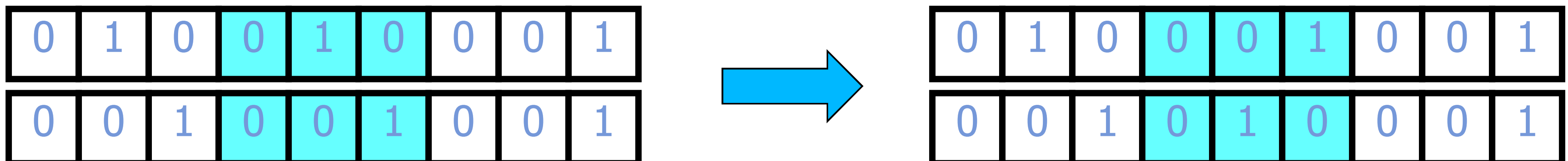
Step V – Search-based Concept Location

- We use a search-based optimization technique based on a GA to split traces into segments that, we believe, represent concepts
- Representation: a bit-vector where 1 indicates the end of a segment



- Mutation: randomly flips a bit (i.e., splits or merge segments)

- Crossover: two-points



- Selection: roulette wheel

Step V – Search-based Concept Location

- Fitness Function:

$$fitness(individual) = \frac{1}{n} \cdot \sum_{k=1}^{n} \frac{SegmentCohesion_k}{SegmentCoupling_k}$$

- Segment Cohesion is the average (textual) similarity between any pair of methods in a segment
- Segment Coupling is the average (textual) similarity between a segment and all other segments in the trace
- Other GA parameters
  - 200 individuals
  - 2,000 generations for JHotDraw and 3,000 for ArgoUML
  - 5% mutation probability, 70% crossover probability

Parallelizing the GA

- Single-computer architecture come from an optimized implementation of our approach.
  - We modified GALib to compute only the fitness values of individuals that have changed between the last generation and the current one.
  - We thus obtained about 30% of computation-time decrease
- The computations were still time consuming
  - Running an experiment with a short trace, containing 240 method invocations (Start–DrawRectangle–Stop), with 2,000 iterations lasted about 12 hours
- Solution: Parallelizing the computations
- Global parallelization:
  - A master computer (client)
  - Several slave computers (servers)
- Client server architectural style, on a TCP/IP network, with 9 servers

Rationale behind the Architectures

- Several individuals share some segments
  - The first two segments of individuals Id1, Id2, and Id5 are identical, as shown below

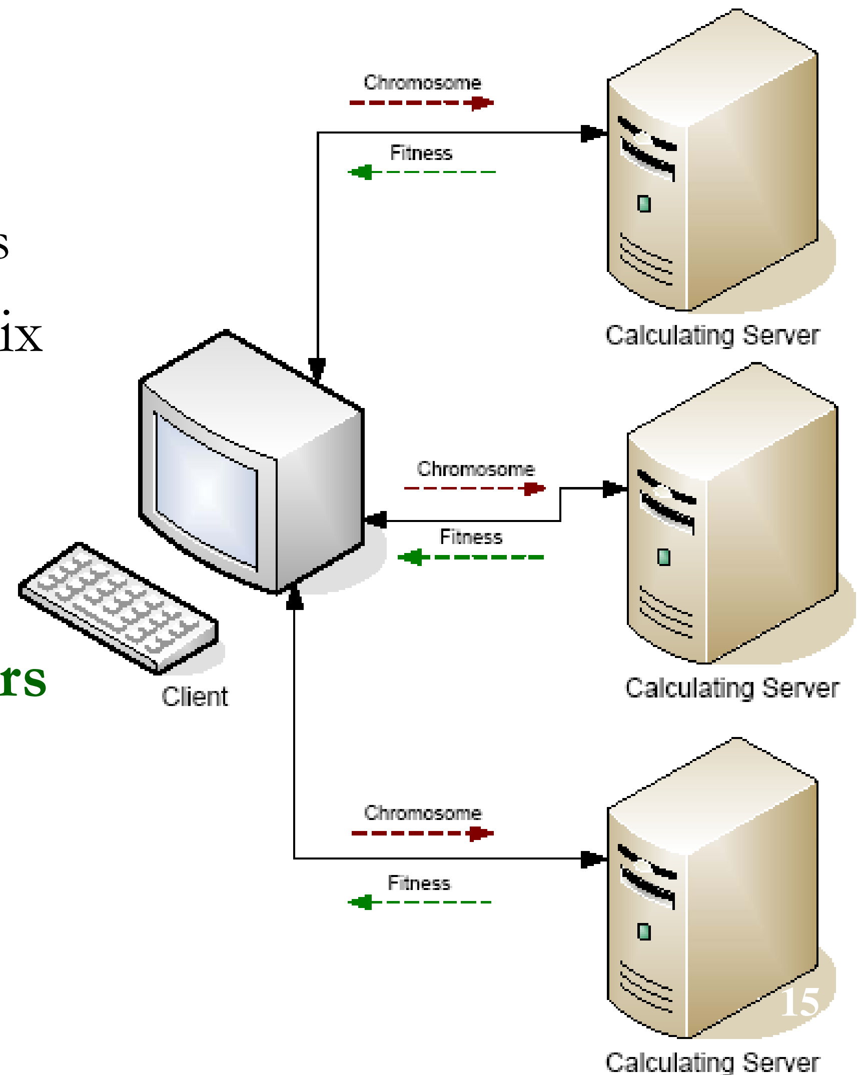
Id1	0001 0001 0000001 0001 0001
Id2	0001 0001 001 0001 0001 0001
Id3	001 00001 0000000001 0001
Id4	00001 0001 0000001 000001
Id5	0001 0001 0000001 0001 001 01

- Once the fitness value of one individual is computed, if segment cohesion and coupling are stored, then they can be reused to compute the fitness values of others

Simple Client Server Architecture

- The servers have
  - No local memory
  - No global shared memory
  - No communication between themselves
- Each server uses its own local LSI matrix

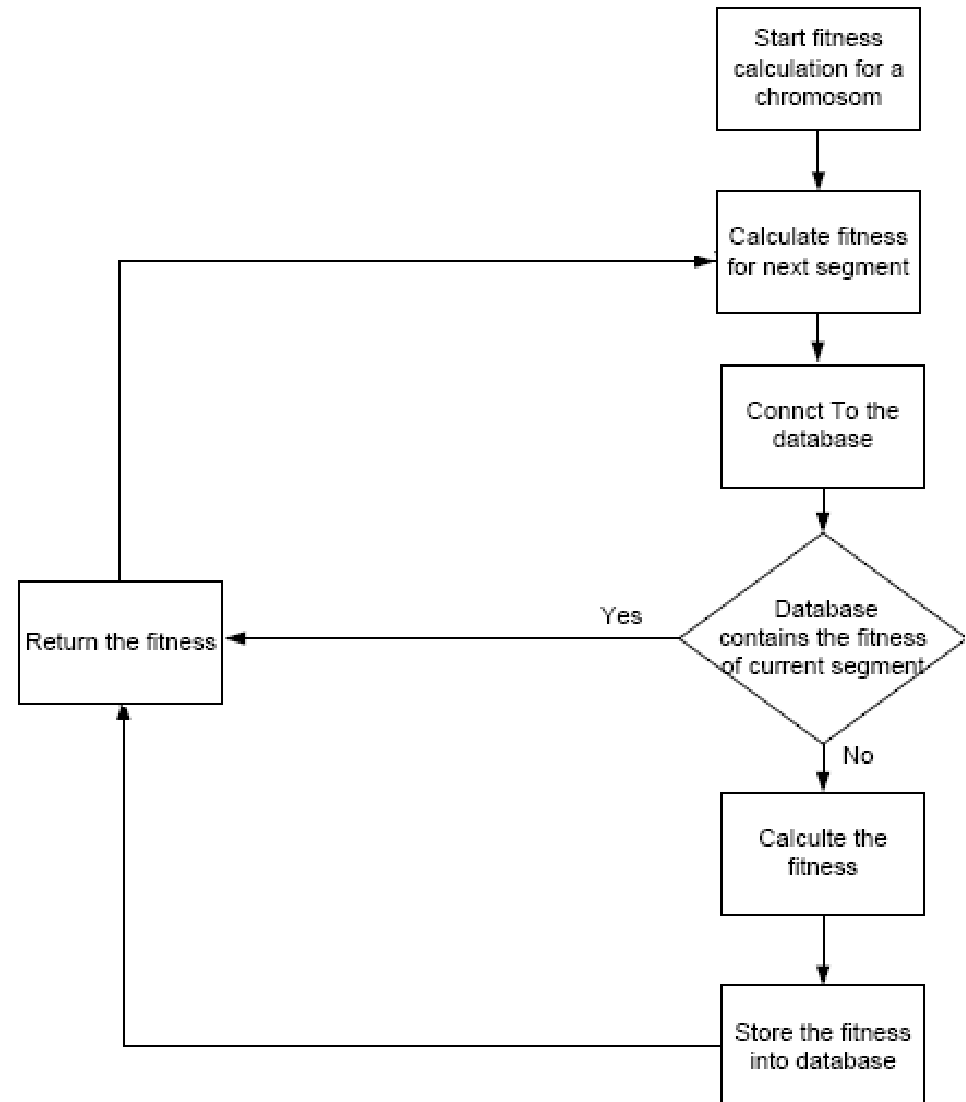
- **In the case of Start-DrawRectangle-stop trace, this architecture reduces the execution time to about two hours**



Database Configuration

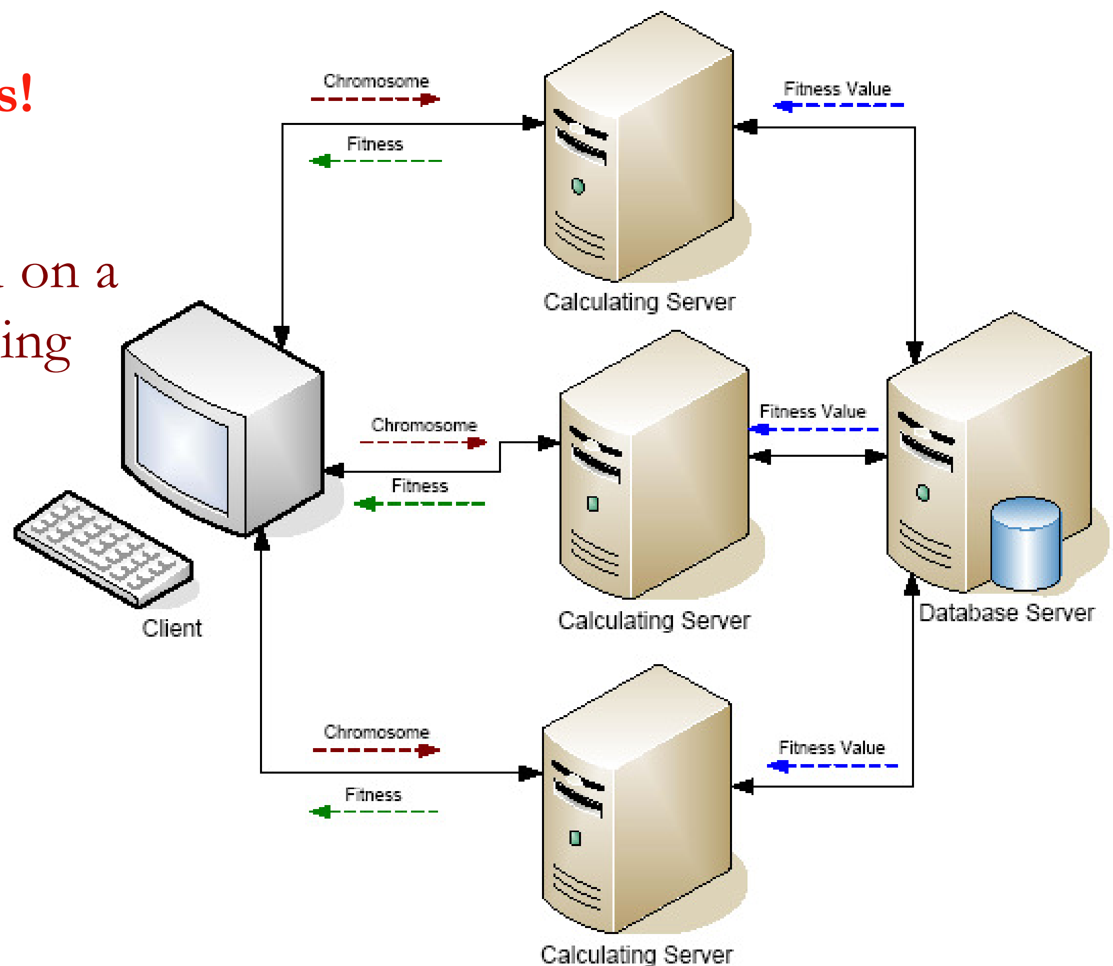
- A large fragment of segments are preserved unchanged in the next generation

- A central storage keeps a history of the previous computations and share the computation results
- No calculation is done more than once
- Each computer benefits from others' computations and does not repeat what has been done once by others



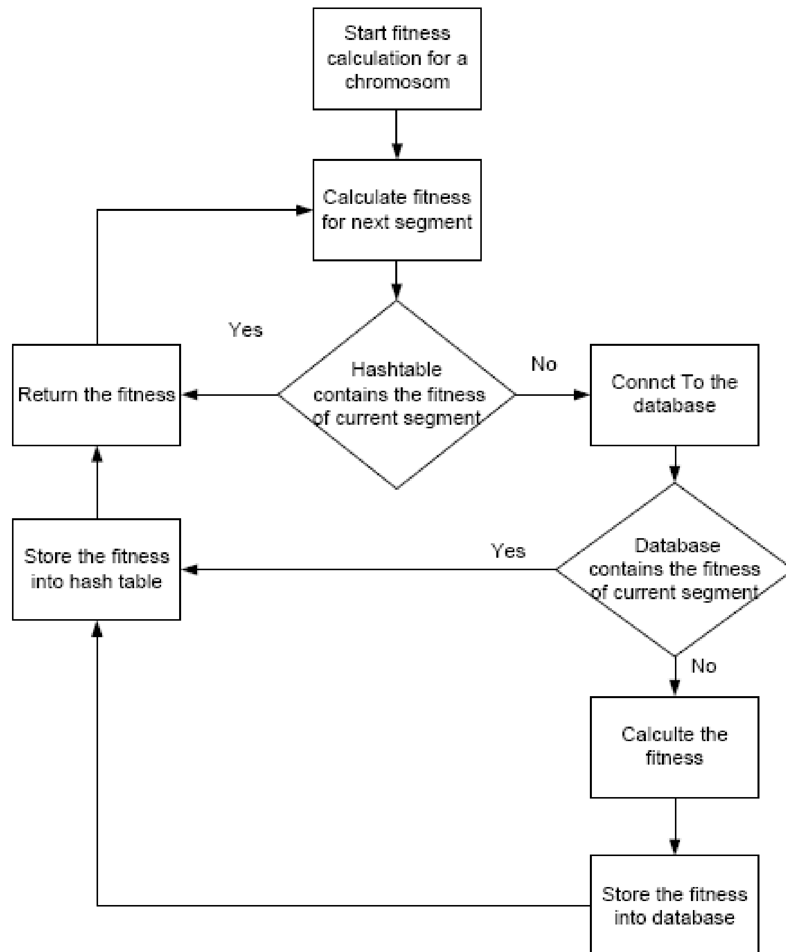
Database Configuration

- **In the case of Start-DrawRectangle-stop trace, this architecture increases the computation time to 13 hours!**
- Accessing to a database located on a different server is time consuming
- Database access issues
  - **Reading, writing, locking**



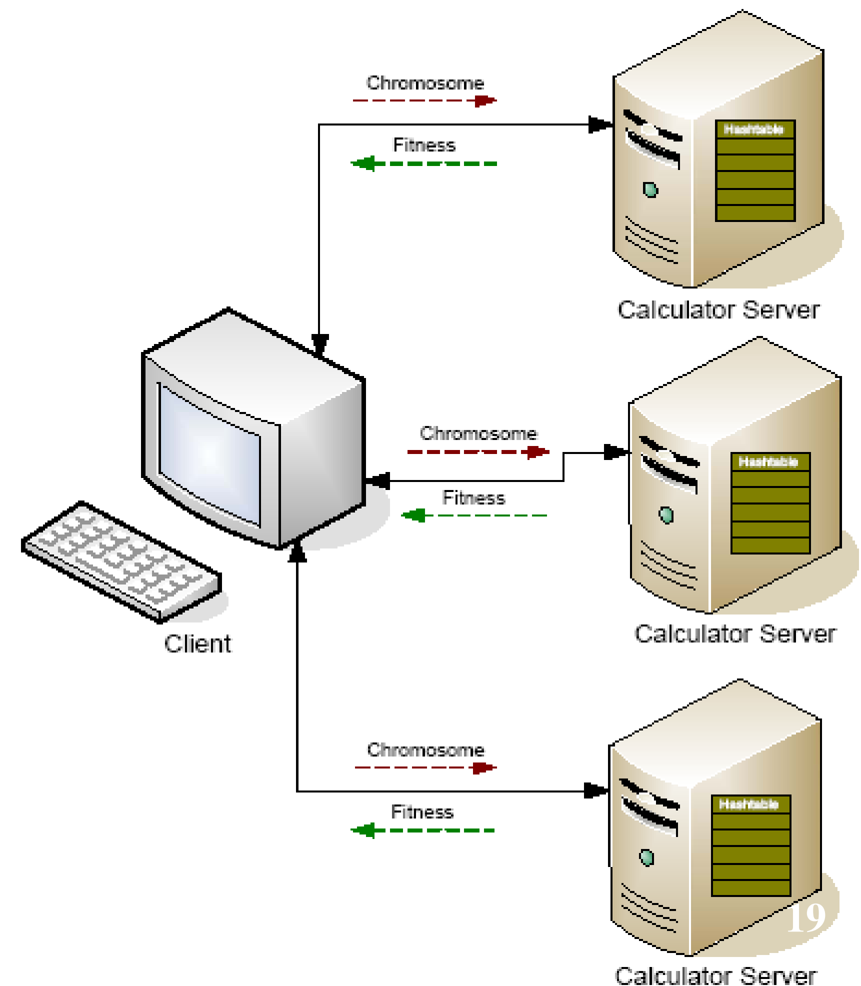
Hash-Database Configuration

- A centralized and local storage are put together to decrease the numbers of accesses to the central database by keeping cached data in a local storage structure
- **Not much better than the last one**



Hash Configuration

- A local storage
  - Only local data is stored in the local hash table of servers.
  - No data is shared among servers
  - Servers only communicate with the client
  - There is no access policy using locking algorithms
  - The access to the already-computed data as well as their storage is efficient
  - It enormously reduces the time.
  - **In the case of Start-DrawRectangle-stop trace, this architecture reduces the execution time to about 5 minutes**



Summery of Results

- Computation time for desktop solution and the distributed architectures with the Start-DrawRectangle-stop trace

Time Measurement			
Architectures	Runs #	Measures	Average
Desktop	1 2 3 4 5	12:09 h 11:39 h 12:21 h 11:50 h 12:38 h	12:07 h
Client–server	1 2 3 4 5	1:44 h 2:36 h 1:53 h 1:40 h 2:13 h	2:01 h
Database	1 2 3	16:36 h 15:3 h 9:52 h	13:50 h
Hash Table	1 2 3 4 5	5:13 m 5:19 m 5:20 m 5:27 m 5:10 m	5:17 m

- Computation time for desktop solution and a hash-table distributed architecture with the Start-Spawn-Window-Draw-Circle-Stop”trace

Time Measurement			
Architectures	Runs #	Measures	Average
Desktop	1 2 3	45:38 h 41:28 h 45:07h	44:07
Hash Table	1 2 3	7:21 m 7:21 m 7:32 m	7:24 m

Conclusion

- We presented and discussed four client–server architectures conceived to improve performance and reduce GA computation times to solve the concept location problem
- We discovered that on a standard TCP/IP network, the overhead of database accesses, communication, and latency may impair dedicated solutions
- In our experiments, the fastest solution was an architecture where each server kept track only of its computations without exchanging data with other servers
  - This simple architecture reduced GA computation by about 140 times when compared to a simple implementation, in which all GA operations are performed on a single machine

Future Work

- We intend to experiment different communication protocols (e.g., UDP) and synchronization strategies
- We will carry out other empirical studies to evaluate the approach on more traces, obtained from different systems, to verify the generality of our findings
- We will reformulate other search-based software engineering problems to exploit parallel computation to verify further our findings

Thank You!

Questions?

Implementation and Environment

- Two traces collected by instrumenting JHotDraw and executing the scenarios Start-DrawRectangle-Quit and Start-Spawn-Window-Draw-Circle-Stop
- Final length of the traces after removing utility methods and compression are respectively 240 and 432 method invocations
- Computations are distributed over a sub-network of 14 workstations
- Five high-end workstations, the most powerful ones, are connected in a Gigabit Ethernet LAN
- Low-end workstations are connected to a LAN segment at 100 MBit/s and talk among themselves at 100 Mbit/s
- Each experience was run on a subset of ten computers: nine servers and one client
- Workstations run CentOS v5 64 bits
- Memory varies between four to 16 Gbytes
- Workstations are based on Athlon X2 Dual Core Processor 4400
- The five high-end workstations are either single or dual Opteron
- Workstations run basic Unix services and user processes
- The client computer was also responsible to measure execution times and to verify the liveness of connections
- Connections to servers as well as connections to the database were implemented on top of TCP/IP (AF INET) sockets
- All components have been implemented in Java 1.5 64bits
- The database server was MySQL server v5.0.77