Bio-inspired Optimization Techniques using Apache Hadoop

Christian Gapp

Institute of Computer Science, University Innsbruck

13.03.2015

Table of Contents

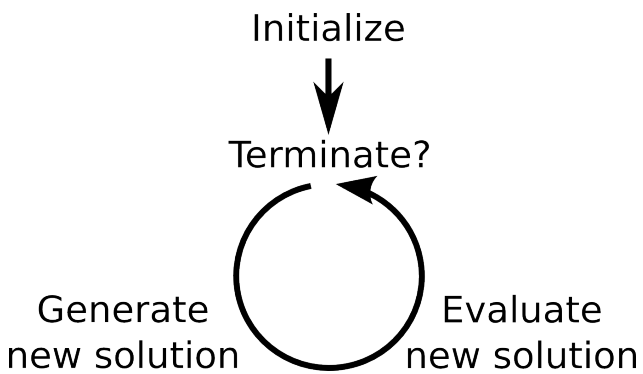
Table of Contents

Motivation

- Era of cloud computing
- Trend goes to Everything as a Service
- Still missing: Optimization as a Service

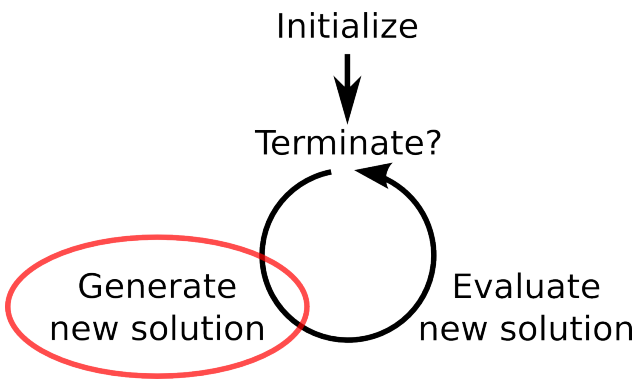
Optimization

- Task of finding minimum/maximum of an objective function
- No exact solving mechanism available for most real-world problems $\rightarrow$ improve solution iteratively



Bio-inspired Optimization Techniques

- Approximation technique: trade accuracy for speed
- Mimic behaviours observed in nature: Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), ...
- Main difference between algorithms: solution generation

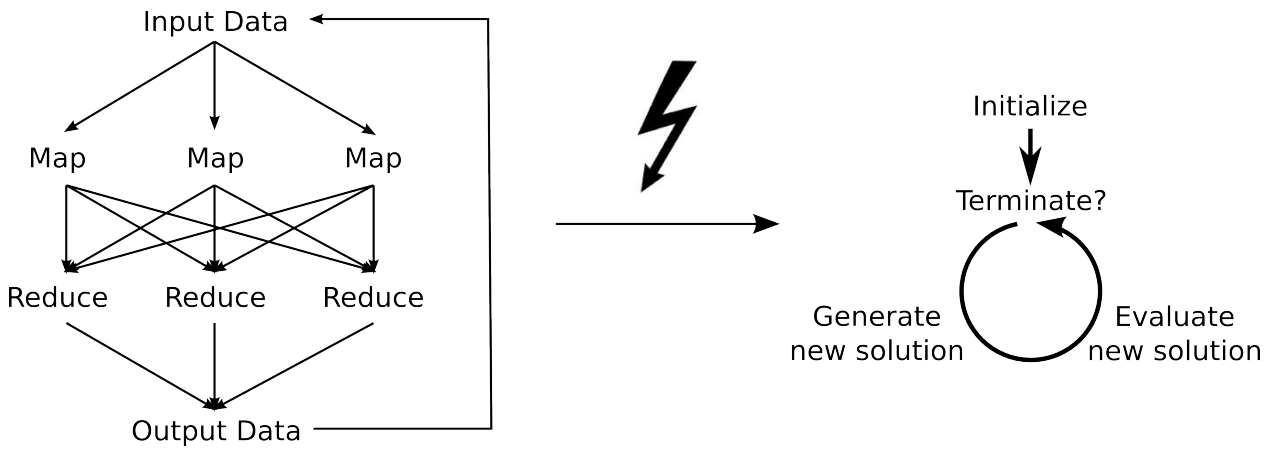


Apache Hadoop

- Hadoop is basis layer for implemented framework
- "OS" for distributed systems
- Suited for cloud usage, runs also on single machine e.g. for development
- Widely deployed and used system (easy to find / switch provider)

MapReduce?

- MapReduce great for problems that can be structured into map and reduce tasks
- Not so good for iterative problems:
  - Data forwarding through file system (slow)
  - Check for termination criteria is outside of scope
  - Computation must be decomposable into map and reduce steps



Better solution

In 2014, Hadoop introduced YARN (Yet Another Resource Manager):

- Flexible usage of resources
- Enables arbitrary computation models
- Gains attraction from developers and cloud providers

Low level, verbose and complex API

Requirements for Optimization as a Service

- Basis layer over cloud → Hadoop + YARN
- Need simple framework for implementation and execution of (bio-inspired) optimization techniques → **Biohadoop**
  - Based on Hadoop + YARN
  - Simple API with support for parallelization
- Interface to interact with Service (not part of this thesis)

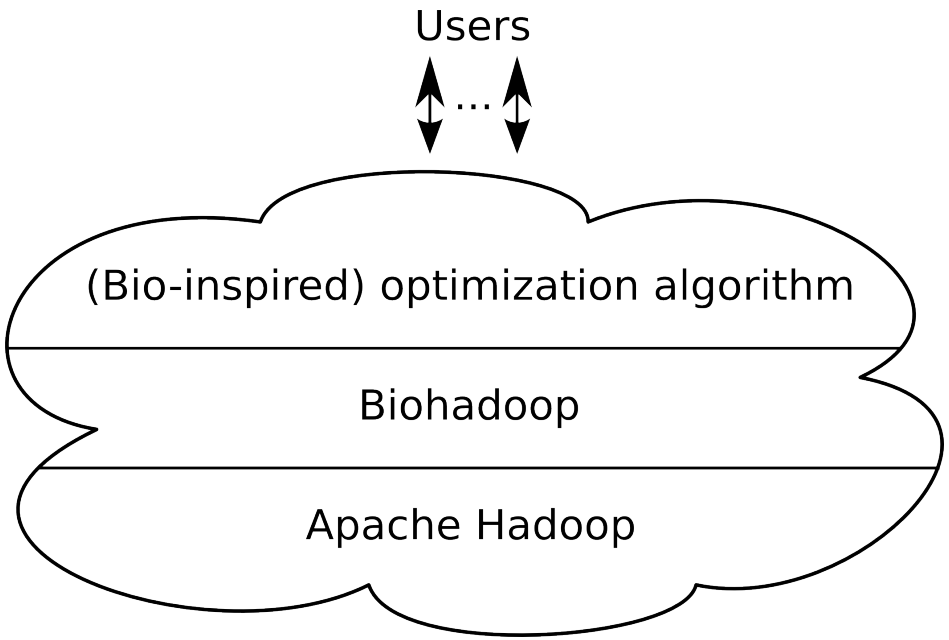
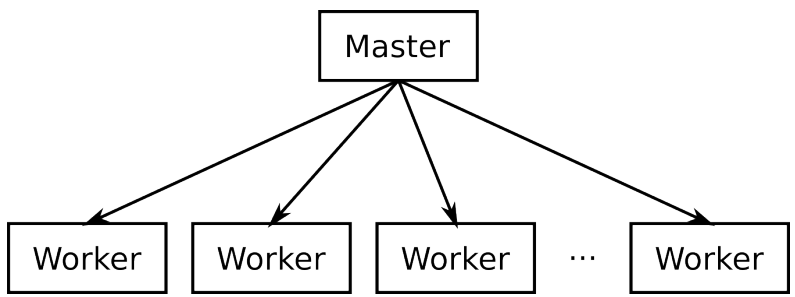


Table of Contents

Biohadoop basic concepts

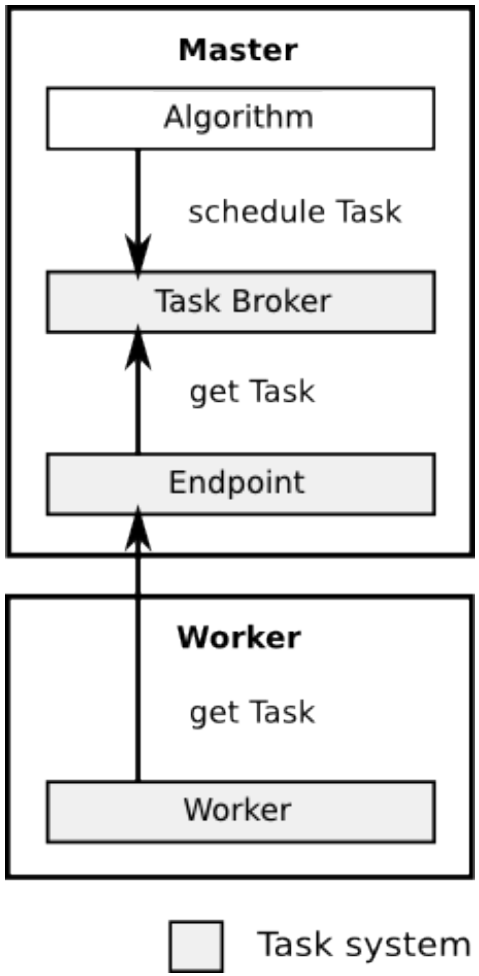
- Master-worker scheme:
  - One **master** executes the main **algorithm** (e.g. GA)
  - Parallel **workers** execute work intense computations (e.g. solution evaluation)



- Work items sent from master to workers are called **tasks**
- Workers compute **results** for tasks and return them to master
- Communication between master and workers is **asynchronous**

System architecture

- Algorithm
  - implements optimization problem and schedules tasks for workers
- Task System
  - TaskBroker queues tasks an passes them to endpoint on request
  - Endpoint is the boundary of the master, workers communicate to endpoint
  - Worker(s) request tasks, compute solutions and return results



Biohadoop usage

Three steps to implement new algorithm:

- Implement `Algorithm` interface, executed on master - typically includes problems main loop
- Implement `Worker` interface, executed on workers - typically includes work intense computations
- Submit tasks from within the algorithm to the task system and wait (or not) for the results

Biohadoop takes care of the rest (Worker startup and shutdown, communication, failure detection)

Example: Genetic Algorithm

- Generate initial population (master)
- Evaluate initial population (**worker**)
- Main loop until termination (master):
  - Generate new solutions from current population (master)
  - Evaluate new solutions (**worker**)
  - Select best overall solutions → current population for next iteration (master)
- After termination, current population is solution to problem (master)

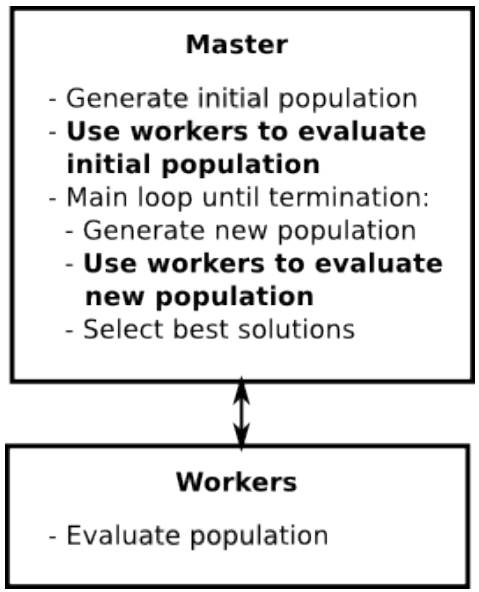


Table of Contents

Cloud Environment

Cloud with 6 identical computers running Hadoop 2.4, each with following specs:

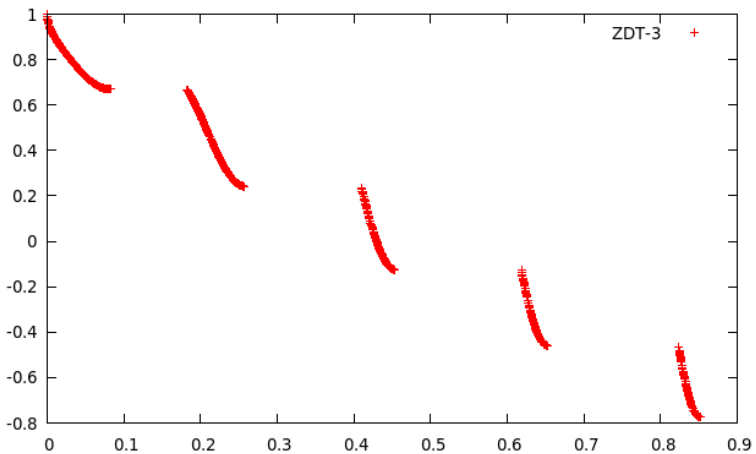
- Intel Core2 Duo CPU E8200 @ 2.66 GHz (2×2.66 GHz, no hyperthreading)
- 6 MB shared L2 cache, 32 KB L1 data cache, 32 KB L1 instruction cache
- 4 GB (2×2 GB) DDR2 RAM @ 667 MHz
- 64 Bit Ubuntu Linux 14.04.1 LTS with kernel 3.13.0-37

Computers connected to 1 Gb Ethernet switch

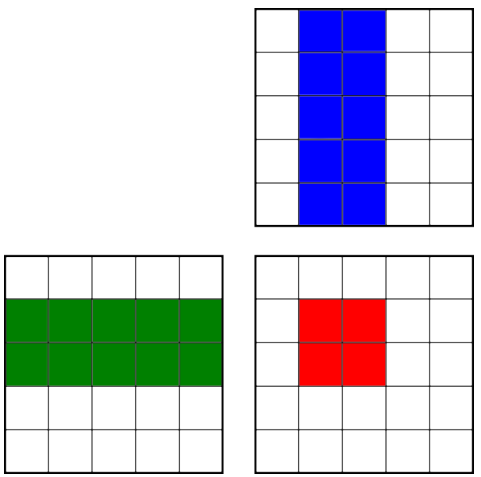
Implemented Algorithms and benchmark problems

- NSGA-II to solve Zitzler-Deb-Thiele's function nr. 3 (ZDT-3): solution evaluation not compute intense
- GA to solve tiled matrix multiplication (TMM): solution evaluation is computational intense

ZDT-3

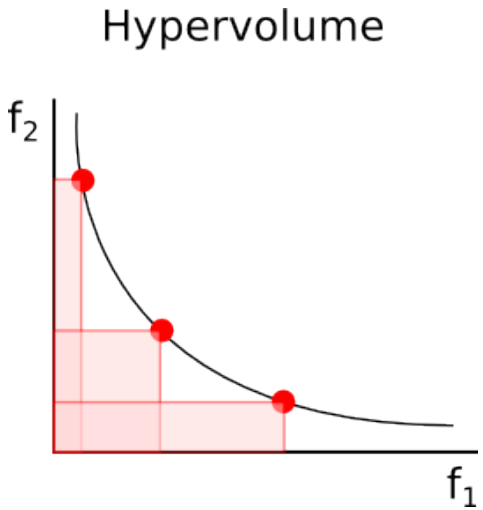


TMM



Correctness of computed results

- Benchmark results were accurate
- ZDT-3: hypervolume as quality indicator showed good results (compared to optimal solution)



- TMM: tiled matrix multiplication provided correct results (compared to simple matrix multiplication)

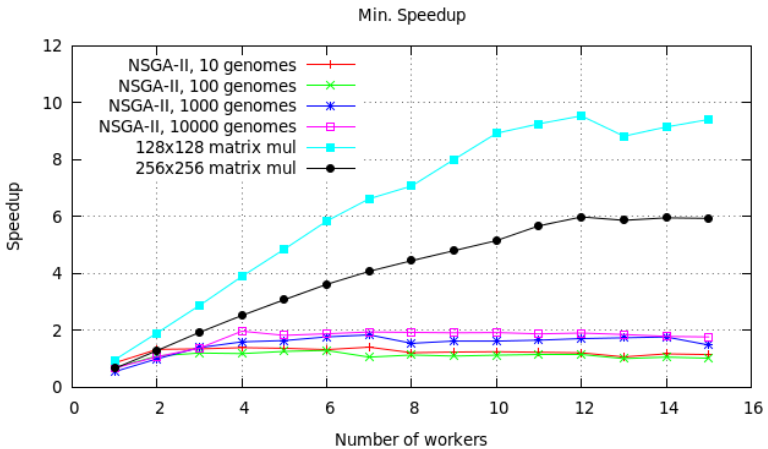
Performance evaluation

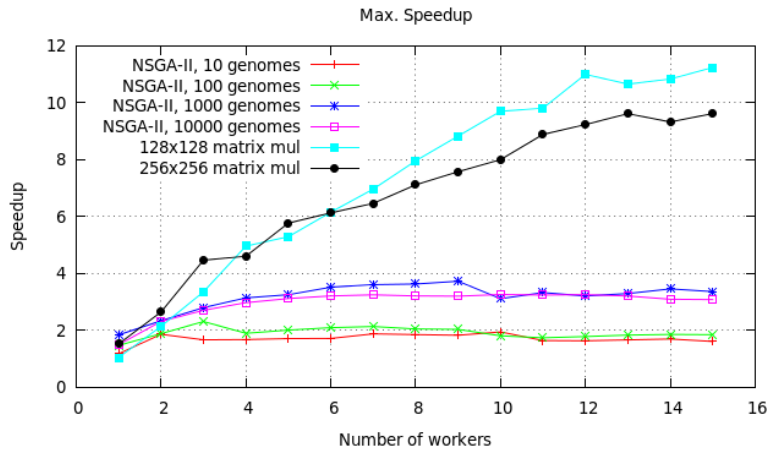
- Measure algorithm execution time for 1 worker $\rightarrow$ basis for performance evaluation
- Increase parallel worker sizes from 1 to 15 to evaluate speedup
- Compare parallel execution times to execution time of standalone, sequential algorithm implementation (no Hadoop, no other framework)

Speedups

Speedups computed using Amdahl's law: $S = T/(T - t_p)$

S = Speedup, T = exec. time for 1 worker, t_p = exec. time for p workers





- Poor ZDT-3 speedups of max ~ 4 for 9 workers
- Good TMM speedups of max ~ 11 for 12 workers

Comparison to standalone implementations

Test Problem	Standalone [s]	Parallel [s]	Performance gain
ZDT-3, problem size: 10	2.852	6.461	0.441
ZDT-3, problem size: 100	2.956	6.128	0.482
ZDT-3, problem size: 1000	7.673	8.330	0.921
ZDT-3, problem size: 10000	71.390	53.706	1.329
TMM, 128×128	132.066	18.386	7.183
TMM, 256×256	1500.705	178.158	8.423

- ZDT-3 performs poor, parallelization has negative impact (except for very big genome sizes)
- TMM profits from parallelization

Table of Contents

Conclusion

Biohadoop is suitable framework to provide Optimization as a Service:

- Runs in the cloud
- Depends only on Hadoop + YARN
- Provides simple API to implement optimization algorithms
- Provides correct computation results
- Good speedups achievable for parallel algorithms (although speedups depend on problem)

Open topic: simple interface to interact with Service

Thank you for your attention

Sources:

- `https://hadoop.apache.org`, last access: 05.03.2015
- `http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html`, last access: 05.03.2015
- X.-S. Yang, Nature-inspired metaheuristic algorithms. Luniver press, 2010
- `https://github.com/gappc/biohadoop`, last access: 05.03.2015

Single Objective Optimization (SOP)

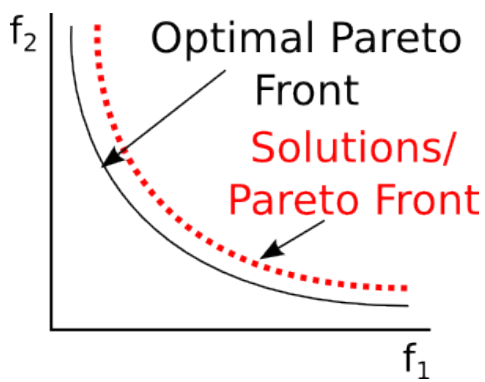
- Optimize for one objective (goal)
- Search for global best solution
- Desired solution: global best

Example: Traveling Salesman Problem (TSP) with 5 cities, 1-5
Objective: find shortest path
Result: [2,5,1,4,3]

Multi Objective Optimization (MOP)

- Optimize for several conflicting objectives (goals)
- No single "best" solution, instead number of solutions to chose from
- Desired solution: Optimal Pareto Front

Example: Traveling Salesman Problem (TSP) with 5 cities, 1-5
Objectives: f_1) find shortest path with f_2) least costs
Result: Pareto Front

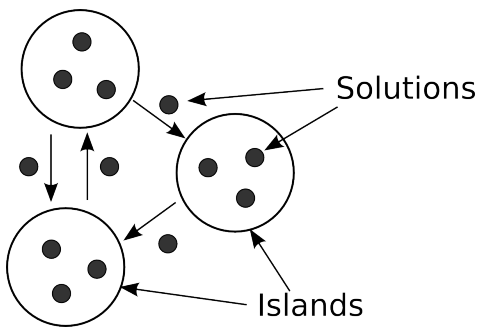


Biohadoop additional capabilities

- Island model (parallelization model)
- Support for workers that don't run on Hadoop. Enables e.g. workers that run in browser or on mobile phone
- Custom communication facilities between master and worker
- Simple file system API
- Simple Config-Builders
- Development mode without Hadoop

Island model

- Multiple independent instances of optimization problems (e.g. GA) executed in parallel, instances are called islands
- Exchange solutions after certain intervals (integrate solutions from other islands)
- Important aspects:
  - When to exchange solutions
  - Which solutions to exchange
  - How to merge solutions



Docker-Biohadoop

- Use of Docker containers
- Pre-build Hadoop environment
- Cluster simulation on single machine
- Play with Biohadoop
- `https://github.com/gappc/docker-biohadoop`, last access 05.03.2015

Biohadoop Java example

Algorithm

```
public class ExampleAlgorithm implements Algorithm {
  @Override
  public void run(AlgorithmId algorithmId, Map<String, String> properties)
    String data = "Hello World!"
    TaskConfiguration<String> config = new TaskConfiguration<>(ExampleWorker.
        class, null);
    TaskFuture<String> future = TaskBroker.submit(data, config);
    String result = future.get();
  }
}
```

Worker

```
public class ExampleWorker implements Worker<String, String, String> {
  @Override
  public String compute(String data, String initialData)
    return result + " - returned";
  }
}
```

Unexpected execution time fluctuations

Large execution time differences found for same benchmarks/settings (up to 50 %):

- Hadoop has big influence on execution times: computation location, worker startup time
- Other influences: Network, RAM, CPU caches, Java Just In Time compiler (JIT), Java libraries (e.g. communication), OS...

$\rightarrow$ VERY difficult to find reasons for performance problems in distributed systems - **better tool support needed**

Improvements

- Web based interface to interact with Service
- Accelerate communication (reduce network overhead)
- Implement batch communication
- Provide API for Worker-to-Worker communication (enable different computation models)
- Implement distributed caching
- Provide API for multi-threading workers
- Provide metrics

Web based interface

- Manage authentication and authorization
- Provide interface to:
  - Upload Biohadoop based algorithm
  - Upload data
  - Retrieve results
  - Start/stop/monitor executions