

# USING A GENETIC ALGORITHM TO TUNE OPENMOSIX LOAD BALANCING PARAMETERS



Alan Ritter, Michael Meehan

Computer Science, Western Washington University

## Abstract

Our goal is to make improvements to the openMosix load balancing algorithm by parameterizing various constants, then searching this space for a configuration which maximizes performance. A *Genetic Algorithm* (GA) has been implemented using a fitness function which measures throughput. As this work is ongoing, we intend to add more parameters to the search space, and incorporate more information (e.g. network traffic and response time) into the performance measure.

## openMosix Background

- openMosix is an open source version of the MOSIX[1] distributed operating system.
- It is distributed as patch to the Linux kernel, and a set of user-level utilities.
- Provides transparent process migration. Processes are automatically migrated to a node where resources (such as CPU time and memory) are available.
- A process' system calls are routed back to its home node over the network as necessary via Remote Procedure Calls.

## Parameters

So far 4 hard-coded constants in the openMosix kernel's load-balancing algorithm have been made dynamically adjustable to user-level processes. This is done through the Linux `/proc` filesystem:

**Amount of Hearsay Information** Each node sends its current load information along with the load information for other nodes it knows about. We vary how much extra information is sent.

**Load Vector Size** The amount of load information maintained by each node. If this parameter is too small not enough information is available to make good migration decisions. If it is too large, then stale information is used.

**Load Reporting Frequency** Each node sends load information every  $\frac{MF}{100}s$ . If load information is sent too frequently it will be unreliable.

**Number of nodes reported to** How many nodes are load information sent to at a time?

## Fitness Function

To evaluate the fitness of each hypothesis, we set its parameters on every node in the cluster, then `fork` off a large number of processes on a single node. The better the load balancing algorithm performs, the sooner the processes will complete their computation. The hypothesis' fitness is thus the reciprocal of the sum of the time for each of the  $n$  processes to complete:

$$\text{fitness} = \frac{1}{\sum_{p=1}^n T_p}$$

## The openMosix Load Balancing Algorithm

The load balancing algorithm in MOSIX is:

**Transparent** Processes are migrated without the user being aware. Parallelism can be achieved by simply `forking` off multiple processes. Existing programs can benefit from parallelization in openMosix without any modification.

**Probabilistic** Every second each node sends its current load to two randomly selected processors.

**Decentralized** There is no central point of control. All process migration decisions are made between a pair of computers. This increases scalability, and fault tolerance.

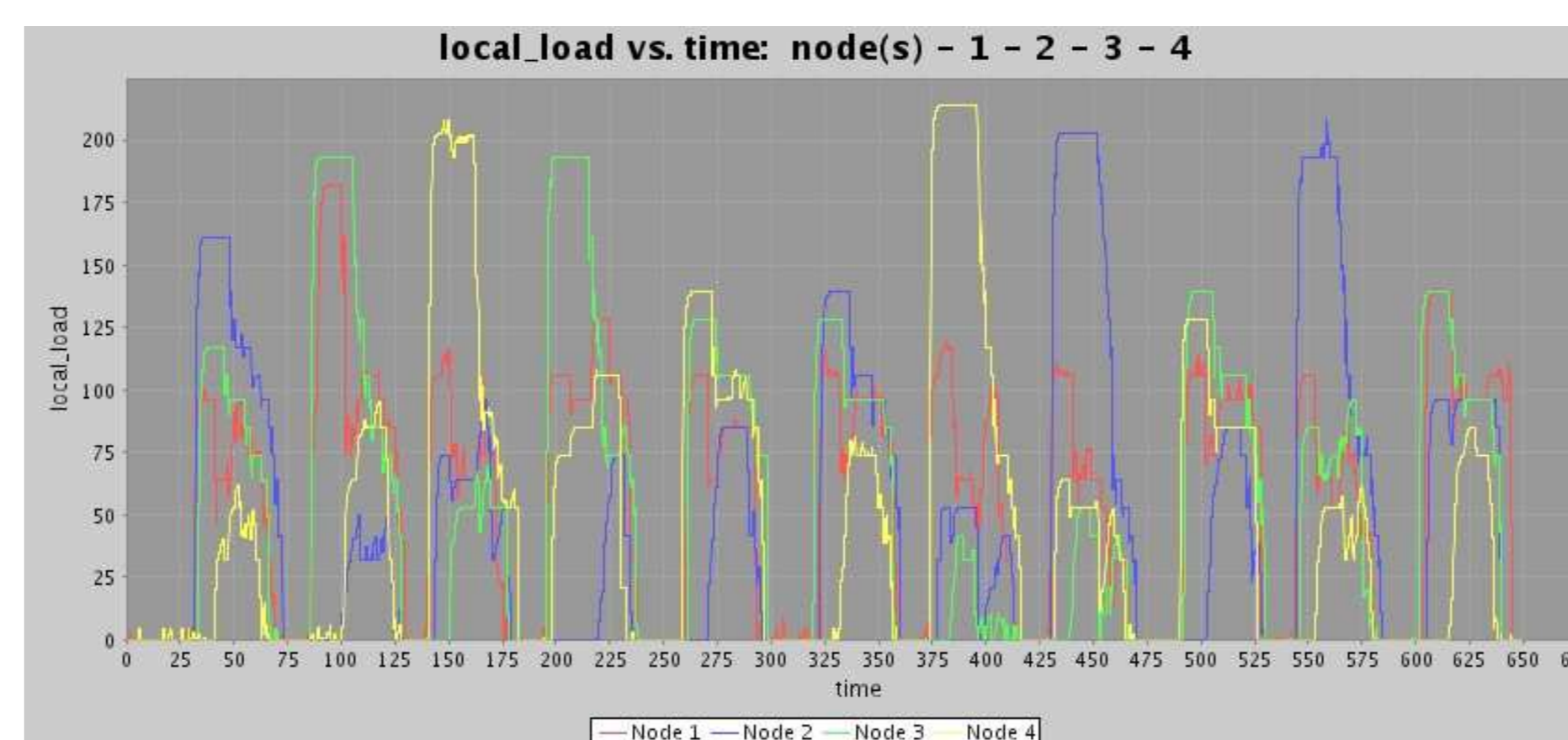
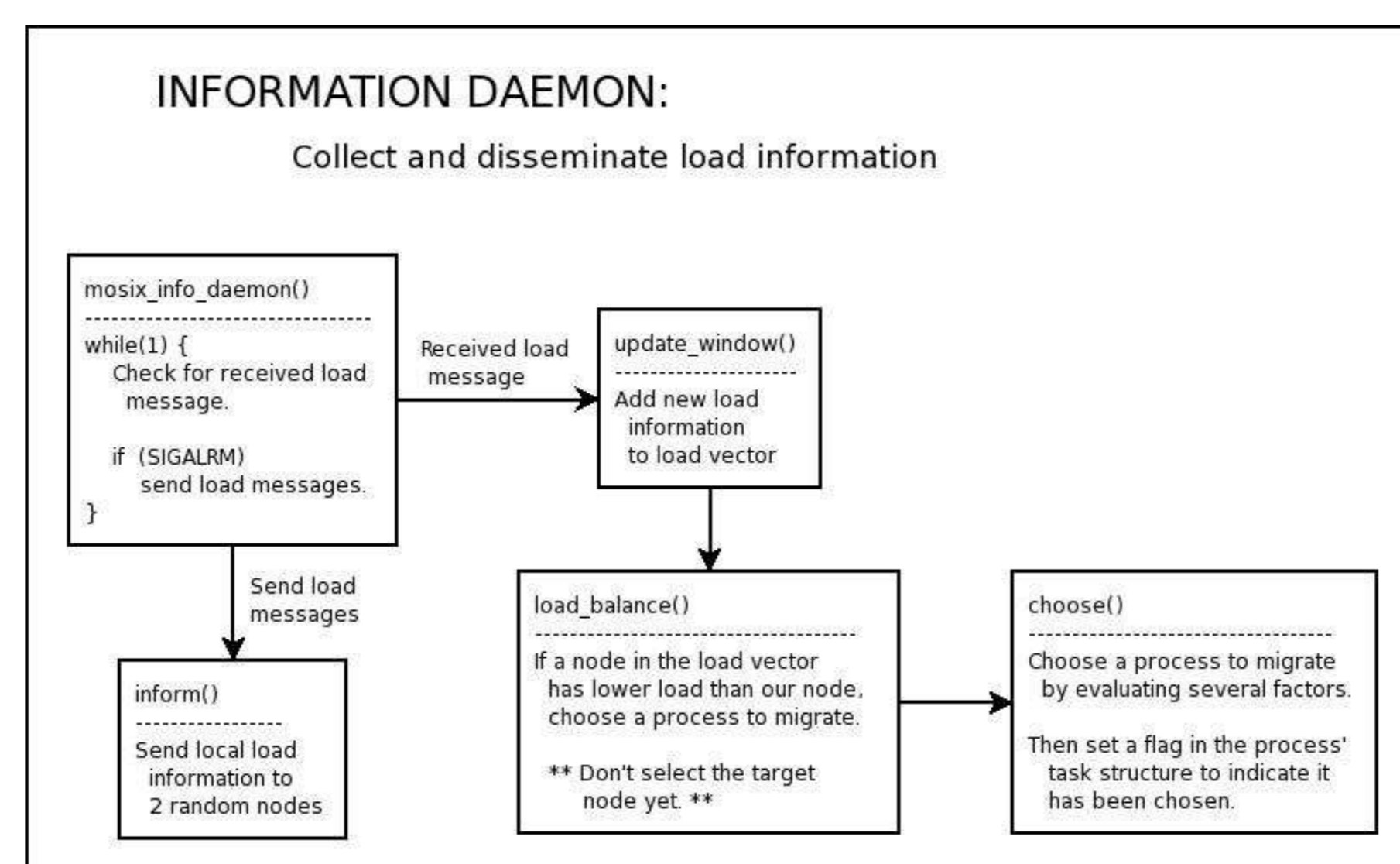


FIGURE 1: A series of load spikes with random parameter settings

## Preliminary Results

Preliminary tests were done on an openMosix cluster of 11 nodes connected via 10MBit ethernet. The fitness function was evaluated using a synthetic load of 50 processes doing 1,500,000,000 multiplications each. This took on average about one minute to complete on the 11 node cluster. The computation needs to be done for each hypothesis, so the population sizes were rather limited. If the test is too short, the system never has a chance to recover from the load spike, and there is too much variance in the fitness function. The results are displayed in figure 2.

## Genetic Algorithms

Genetic Algorithms (GAs) are useful for finding approximations to optimization problems. They maintain a population of current hypotheses (set of parameter values), which are evaluated using a *fitness function*. The initial population is randomly generated, and successive generations are produced by applying the following operations:

**Selection** Only the most fit members of a population are maintained. This corresponds to the idea of "Natural Selection".

**Crossover** Hypotheses (members of the population) are combined to produce a new hypothesis which has characteristics of both parents.

**Mutation** A small number of hypotheses are randomly modified.

## THE GENETIC ALGORITHM

```

Generate a random population, P
while |P| > f do
  Evaluate the fitness of each m in P
  Write each m in P to the log file
  Crossover(P)
  Mutate(P)
  Remove the r|P| least fit members from P
end while
    
```

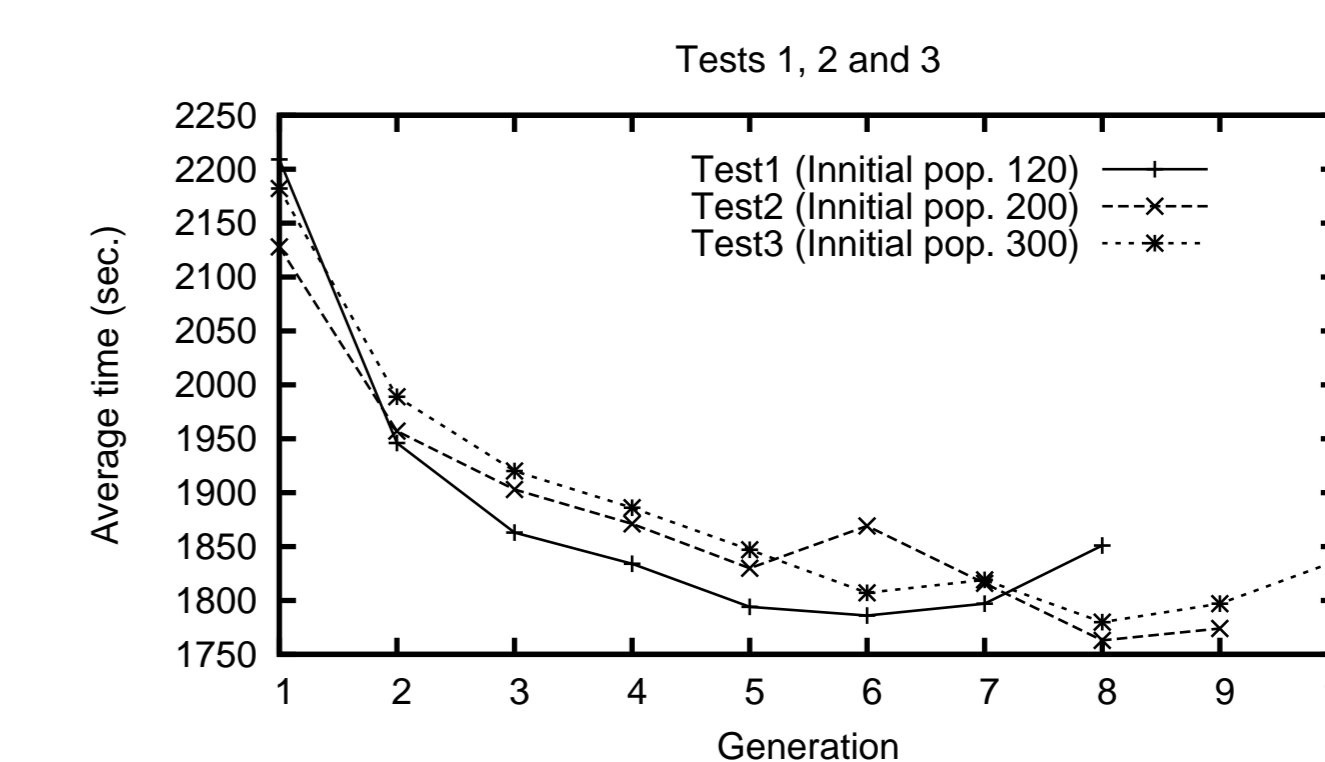


FIGURE 2: Average total time for each task to complete on an 11 node cluster

As you can see, the time needed to complete the load generally decreases at each successive generation. This is because the less fit hypotheses are gradually replaced in favor of those which are more fit, as we would expect.

For tests 1 and 3, the average fitness begins to increase during the last two generations. This is most likely because the population size is decreased too quickly, or the mutation and crossover rates are too high at this stage. In addition, because the load balancing algorithm is probabilistic, there is a large amount of variance involved in evaluating the fitness function. A larger population size offsets this effect, whereas for smaller populations the results are less reliable.

## References

- [1] Barak, Guday, Wheeler, The MOSIX Distributed Operating System, Springer, Verlag, 1991.