Winter Capstone Term Paper
University of Washington Bothell

# Evaluating Repast Simphony for Agent Based Geometric and Combinatoric Simulations

Maxwell Wenger
*mdwenger@uw.edu*

Supervised By
Dr. Munehiro Fukuda

March 22, 2021

# Abstract

The purpose of my research project is to compare the performance and programmability of different tools that may be used for geometric and combinatoric simulations. Repast Simphony[1] is one of many products and paradigms we will be evaluating. The others products we are evaluating are as follows:

- MASS[2], an agent-based spacial simulation suite developed by our team at the Distributed Computing Laboratory at the University of Washington Bothell.
- JClik[3], a multithreading tool kit for Java. The evaluation for JCilk has been completed by Jonathan Acoltzi.
- IBM Aglets[4], which is a mobile agents platform for Java.

The performance of all of these tools will be evaluated on multiple axises. Statistics will be gathered about how each tool performs based on number of computing nodes, and the size of the simulation. The programmability will be evaluated on the amount of code, ratio of code to "boilerplate" code, and how fitting the paradigm was for the algorithm.


[1] https://repast.github.io/

[2] https://depts.washington.edu/dslab/MASS/

[3] http://supertech.csail.mit.edu/jCilkImp.html

[4] The original site is no longer online, but a mirror is being hosted at http://alumni.media.mit.edu/~stefanm/ibm/AgletsHomePage/index_new4.html

During the winter quarter, I have completed my analysis of IBM Aglets. This paper will outline the work I had to do to complete this.

# Contents



# 1 Quarter Progress Summary

This quarter I completed the benchmarking programs for IBM Aglets, and have completed an analysis of the performance and programmability of those programs.

# 2 Setting up Aglets

IBM Aglets had many challenges to overcome when setting up Aglets because of its age. My main source for installing Aglets is the paper listed in their documentation.[5]

The first challenge I ran into was of Java dependencies. Java Aglets expects Java 1.2,

[5] https://phoenixnap.dl.sourceforge.net/project/aglets/User_s%20Manual/March%202009/manual031209.pdf

but there was no officially supported Centos 7 package for Java 1.2. After trying a few alternative repos, and trying old centos packages on centos 7, I finally ended up needing to go through the oracle archives to build java 1.2 from source. Because the hermes machines are running on nfs, it was fairly simple to do this fix once and move it to all the other hermes machines once it was setup on one.

The next challenge I ran into was that Aglet's CLI was not sufficient for use as a master node to start and manage the Aglets. Agelts was meant to be managed from their GUI built into the program. The hermes machines are headless, meaning I couldnt simply remote desktop into the machines. I ended up setting up X11 forwarding over SSH from the hermies machine to my desktop so that I could start the X11 server on hermes, launch the aglets GUI inside of the X11 server, and forward that to my X11 client on my desktop. This gave me the GUI on my desktop for me to use. The other hermes machines did not require this configuration as the CLI was sufficient for the slave nodes.

# 3 Working with Aglets

To begin writing programs with aglets, I used *Programming and Deploying Aglets*[1]. This was the best and only resource I found. There was very little support or references I was able to find outside of this book and the original documentation.

Actually creating an aglet and moving it around was actually quite simple to do. But, being able to communicate between aglets or recall aglets was very difficult. It seemed that once I dispatched an aglet to another machine, I was unable to receive message replies or recall the aglet. Even using the examples in the book verbatim resulted in serialization issues that I could only attribute to things broken inside of the aglets library.

My first solution was to attempt to reimplement the asynchronous messaging library that was not working in aglets, but even getting synchronous replies were having the same issue. This lead me to believe that the issue lied in the internals of aglets rather than just the message wrapper.

I then went on to building my own messaging protocol for the aglets to communicate. As these aglets were living on servers that had access to the same nfs share, I decided that I could use this to my advantage and do file based communication between aglets over NFS. So, I had each aglet write results to a file, and the master aglet would monitor this folder on NFS and read in the results.

This file-based communication was functional, but it had very inconsistent performance. Swinging between very performant and very slow. This would not be a viable way of measuring performance because of this.

I then implemented a socket-based communication platform. This relied on a socket broker that each aglet would report to via socket communication. The following is an outline of how this system worked:

1. The broker must be started.
2. The master aglet would generate a unique ID, and register the number of

workers, aglets, and details about the run with the socket server, using the ID to uniquely identify the run.

3. Once the broker recieves this message, it creates a new run and adds it to the current run list. It also starts a timer here.

4. The worker aglets are dispatched, and as they finish their runs, they send a socket message to the broker reporting that they have completed their run and reports details about their run.

5. Once all aglets report a completed run, the broker will end the timer, and log the run to a file.

This socket based communication, although not pure software agent form, was the most consistant way to implement these programs and was used to evaluate aglets. It was also a very nice tool to use during performance evaluation as I added support to check if a run is complete based on ID, so the master node was able to know when the current run is done. I used this to my advantage by setting up batch runs where I could loop through many configurations and automatically gather performance statistics.

# 4 Results

The most interesting thing I found from this work was there was not a direct correlation between more hosts or more aglets to performance. The performance statistics formed a

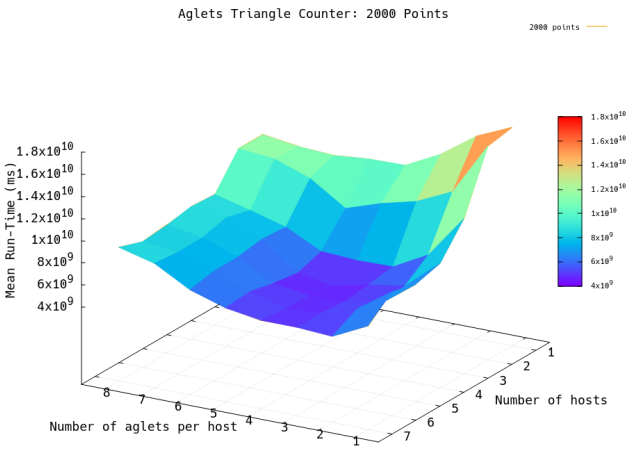


Figure 1: Mean runtime of triangle counter on aglets with 2000 points

parabolic curve many times where the most performant configuration is an optimal number of hosts and nodes.

Figure 1 shows how adding more and more hosts or aglets does not necessary make it faster. Too many aglets increases the overhead, but not enough does not take full advantage of parallelazation. Same with hosts. The best results is a mix of many hosts and many aglets. In the future, I would like to explore ways of having intelligent agents determine the optimal number of hosts and aglets on their own.

# 5 Conclusion

Although aglets proved to be extremely challenging to work with, I took a lot of value and enjoyed implementing a broker service with a socket-based API for the aglets to use. In

addition to that, I really enjoyed seeing the relationship between hosts and nodes during evaluation, and it has raised many questions about these types of systems that I want to explore further.

# References

[1] D. Lange and M. Oshima, *Programming and deploying java (TM) mobile agents with aglets (TM)*. Boston, MA: Addison Wesley, 1998.