A Human Behaviour Model Agent for Testing of Voluntary Computing Systems

MARIUSZ MATUSZEK
Mariusz.Matuszek@eti.pg.gda.pl

Gdansk University of Technology
Department of Computer Architecture, Faculty of ETI
11/12 Narutowicza Street, 80-233 Gdansk, Poland

Abstract: This paper presents a design and performance of a voluntary-based distributed computing system testing agent, implementing a human behaviour model. The agent, nicknamed *iRobot*, was designed and implemented to enable controlled, large scale testing of core algorithms of Comcute – a new voluntary distributed computing platform complementary to BOINC. The main agent design goals were: emulation of human behaviour when browsing web pages, flexibility of configuration, ease of use and ability to run multiple agent instances on a single host. Opening paragraphs of this paper explain the motivation for the design of the agent, and context in which the design of the agent was located. Then the design is described in more detail. Finally, performance data and example configuration is given, with closing remarks focusing on possible extensions and improvements.

Key–Words: distributed system performance testing, voluntary computing, human behaviour model, agent modelling, human computer interaction, intelligent agents.

1 Motivation

Supercomputing power availability is at the core of modern science and the demand for it is ever growing. Besides traditional centralised supercomputing, a lot of effort is being spent on tapping into the distributed computing resources of Internet connected personal computers. BOINC [3] is the most known and successful platform of this kind. The approach taken by BOINC and other similar projects requires installation and configuration of a dedicated client software on volunteers' computers, which may pose a difficulty (both trust and skill wise) to some users. Also, in BOINC model, volunteers are encouraged to keep the client software running for extended periods of time, which is well matched to stationary personal computer use cases, but raises issues when trying to tap into computational resources of mobile platforms. To address both issues, another approach to computational resource sharing was proposed, where the only user action required to share their computing resources is pointing a web browser at a specific web address and clicking a mouse on a confirmation link[1]. This approach was implemented and is known as the Comcute Project [4].

It should be observed, that patterns of volunteer computing power availability differ in both approaches. In BOINC-like cases it can be expected that, once connected, computers will stay connected and available for extended periods. In Comcute, due to ease of computing power sharing and mobility of some users, the pattern can be described as 'compute as you go' and closely resembles users' web browsing patterns, resulting in frequent connections and disconnections. Those differences required careful addressing in the design of core algorithms, as well as a design and development of a dedicated testing tool based on a concept of human behaviour modelling agent, which would mimic patterns of Internauts' behaviour and allow to test Comcute core in a controlled environment.

The design and performance measurements of the Comcute System were presented elsewhere [5, 6, 7]. This paper presents the simple yet versatile human behaviour modelling agent, nicknamed *iRobot*[2], used to test the implementation of a Comcute system. The diagram of main components of test environment, where iRobot agent was deployed, is presented in Figure 1.

[1] From the technology point of view, just visiting a page is enough to engage the resources, but this raises legal issues.

[2] In memoriam of Isaac Asimov.

ISBN: 978-960-474-381-0

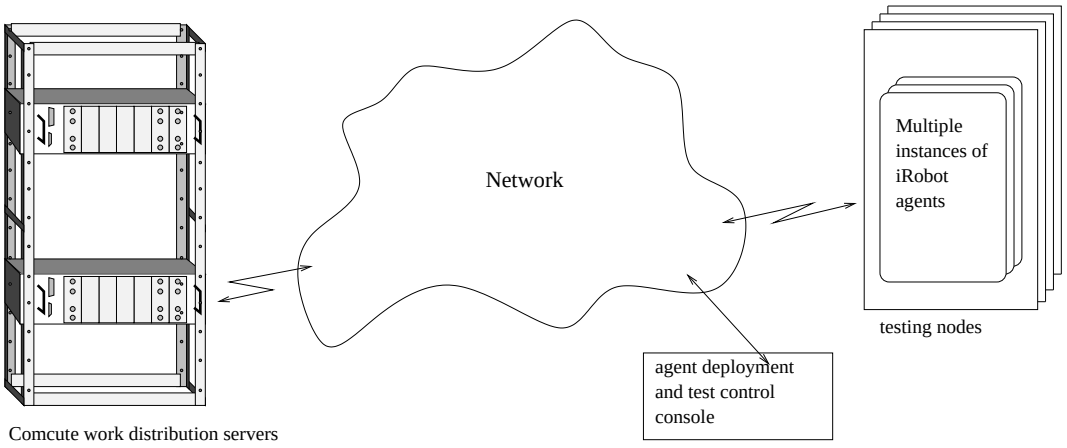

Figure 1: Deployment of iRobot agent related to main components of testing environment

2 Related work

While there are many tools designed for testing performance of task coordination in distributed computing [8, 9], they mostly focus on technical metrics, like throughput, communication delay, number of jobs coordinated, etc. This fits well with the traditional model of distributed computing, and, to some extent, also volunteer based computing using the BOINC approach, where computers stay connected and available for computations for relatively long periods of time. Also the considerable amount of research into voluntary computing focuses mostly on BOINC-like arrangements [10, 11, 12]. None of the tools considered for the task of testing the 'compute as you go' approach implemented in the Comcute environment really fit the requirement to model the behaviour of the Internauts. Therefore a decision to design iRobot was made. The use of intelligent agents to mimic human actions in computer systems is a well established approach [1, 2].

In the course of the Comcute project measurements of the Internauts behaviour were carried out, in order to establish values for metrics like average frequency of visiting a page and average time of stay on a page, with associated variances [13]. These values were later used in implementation of the iRobot agent.

3 iRobot agent design

iRobot was designed to be lightweight, modular agent, implement an Internaut behaviour model and to be flexible in use, including ability to be remotely controlled. The way these design criteria were addressed will be described in the following paragraphs. The internal architecture of the agent is shown in Figure 2 and referenced in following descriptions.

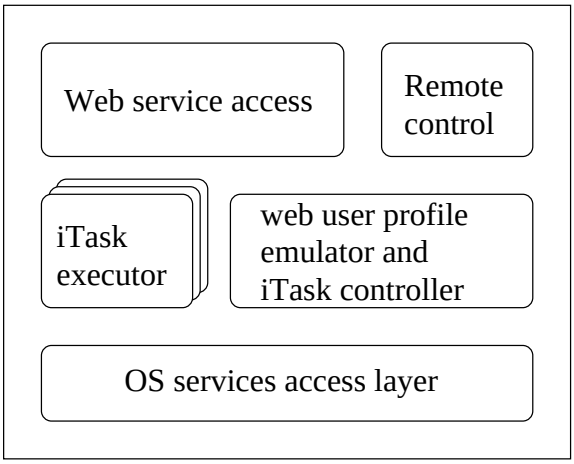

Figure 2: iRobot agent architecture

3.1 Modularity

One of the design goals of Comcute was to utilise the offered computing power in the best way possible and match the task sent to the capabilities of the computer volunteering the resources. In order to do this, a loader code, run when the volunteer clicks the 'join computations' link, reports to Comcute the capabilities of the computer available from within the web browser. This allows to select, for example, a Java applet computational module instead of Silverlight or Flash ones, if it was determined, that on this particular platform Java applets were the most efficient.

iRobot agent was designed to run any web technology, for which there is a runtime environment available from the command line. The information about available technologies is entered into the iRobot by the person preparing the test environment, by means of enabling iTask executor modules (see fig. 2) for specific technologies. Once this is done, iRobot will send an JSON-encoded object identifying

the computational capabilities present, to the Comcute edge task server (an 'S' server in the Comcute project nomenclature) which will cause it to respond with the best matching task (or with no task if no match between task and technology can be made). Once a task is received, a separate process is spawned (named iTask) and its image replaced with an interpreter suitable for the chosen task technology. This task is then being supervised by the iTask controller (fig. 2), which performs actions on iTasks according to the Internaut human behavioural model described in the next section.

3.2 Internaut behaviour model

During Comcute Project development a research into Internauts web usage patterns was made [13]. Also a number of already present papers on this subject were identified, of which one was particularly applicable [14], as it contained statistical data, which proved helpful in parametrising the usage patterns discovered in [13].

Two important metrics were identified: average time spent by an Internaut on a web page and average time between visits. Both metrics were expressed as random variables with Gaussian probability distribution, with known mean and variance values. Using those values, the iTask controller thread calculates the life time for each iTask being run as well as a time delay between the iTask end and another get task request to the Comcute servers. The task life time T_l is calculated according to the formula 1 and the delay between requests T_d is calculated as given in formula 2.

$$T_l = |f(x)| \quad \text{where} \quad f(x) = \frac{1}{\sigma_l}\phi\left(\frac{x-\mu_l}{\sigma_l}\right) \quad (1)$$

$$T_d = |f(x)| \quad \text{where} \quad f(x) = \frac{1}{\sigma_d}\phi\left(\frac{x-\mu_d}{\sigma_d}\right) \quad (2)$$

In both cases the default values of σ and μ can easily be overridden by the test operator. Once the T_l expires, the process running the current iTask is forcibly killed. This mimics the situation when the Internaut decided to point their browser away from the current page. The core agent behaviour timeline, excluding remote command and control activities, is shown in Figure 3.

3.3 Remote control and task arguments

An important consideration in iRobot agent design was flexibility. This was addressed in two ways. One was the built in ability for remote control. There are

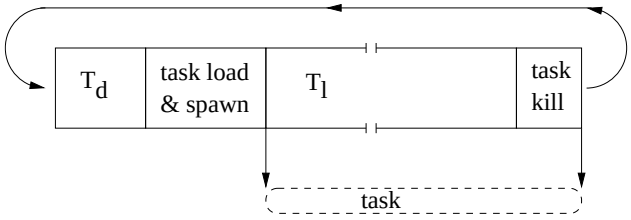

Figure 3: A timeline of the core Internaut user model behaviour of the iRobot agent

two basic modes of iRobot operation: active and waiting. When iRobot starts, a check for existence of two file objects is made. These objects can be either local, specified by a local file system paths, or remote in which case URLs are given. The first object recognised is an 'iQuit' object. Existence of this object will cause immediate abort of iRobot's execution and the program will exit. Once this occurs, new iRobot instances have to be created by the test operator if testing should be continued.

The second recognised object is an 'iRun' object. Presence of this object will switch iRobot to an active state. This is the main operating state of iRobot, in which new tasks are requested and run, according to the behaviour modelled by the iTask controller. After every iTask finish another check for existence of the 'iRun' object is made. If it exists, iRobot remains active. If it does not exist, iRobot switches to the standby mode, unless iQuit object is present in which case iRobot quits immediately.

The second way flexibility was addressed was introducing the ability to specify additional arguments to each task returned by the Comcute servers. These arguments are specified in a text file object 'iOpts.txt' and, similarly to the iRun and iQuit, it can be a local or a remote object specified by a ULR. The format of this file is simple. It consists of groups (paragraphs) of lines, where each group is separated by one or more empty lines. The first line in a group is considered to be a pattern, which will be matched against a (part of) task name returned by a Comcute server. The remaining lines in a group are considered to be argument specifications and are passed verbatim to the matched task in the order of occurrence. Patterns are also matched in order of appearance, till the first successful match. Figure 4 shows the format of an iOpts.txt.

The first group will match against any task containing 'PerformanceTesterJava' in its name and will produce three arguments when such task is run. The second group will match any task with 'PerformanceTesterFlash' in its name and present two arguments to it. Finally the third group will match any task

```
PerformanceTesterJava
arg1 = 100
arg2 = 1000
arg3 = 10

PerformanceTesterFlash
100
2000

manceTester
55
512
200
logging=true
```

Figure 4: Example of an iOpts.txt configuration object. First line of a paragraph is a pattern to be matched against a task name. All remaining lines of a paragraph are arguments to be passed to the task

with 'manceTester' in its name, unless it was matched by earlier groups, and will append four arguments to the task command line.

The 'iOpts.txt' object will be read on every transition from sleeping to active state. This lets the test operator update the task arguments without a need to restart iRobot agents.

4 iRobot agent performance and application

Due to flexibility of iRobot agent it can be used in testing scenarios ranging from brute force load tests to finely tuned simulations of human-generated traffic and resulting patterns of computational power availability. A switch between the two extremes (or anywhere in between) can be easily achieved by the test operator by adjusting the mean and variance of T_l and T_d generators.

The low memory footprint makes it easy to run multiple agent instances on a single computer, making it easy to deploy large scale testing with only a limited set of resources available. During tests performed with iRobot it was observed, that the most limiting factor to the number of possible deployed instances was the memory footprint of actual runtime environments used to run tasks returned by Comcute servers. In one test, using only Java test tasks, it was possible to deploy 69 simultaneous iRobot instances together with simultaneously running test tasks on a single node configured with 4GB of RAM and dual Xeon CPUs. To achieve such result, it was necessary to change (lower) the defaults of Java VM memory allocation, which required only a definition of necessary parameters in the iOpts.txt file. A summary of the agent performance on selected host computers, running Java tasks with default JVM settings, is given in Table 1.

Table 1: iRobot agent performance on selected hosts

host platform	instances
dual Intel Xeon, 4GB RAM	51
Intel i7, 24GB RAM	172

An example result showing a similar test run, with a 64 running iRobots spread across 8 hosting computers, is illustrated in Figure 5. Another example, with 512 deployed iRobots on 8 hosts is shown in Figure 6. Both test runs shown were of the load test type with a limited T_l and short T_d, which resulted in some loss of computations data and almost instantaneous reconnections, as intended. In both cases a reaction of Comcute to loss of data can be observed, near the end of time lines, where a second period of task activity can be observed after rescheduling.

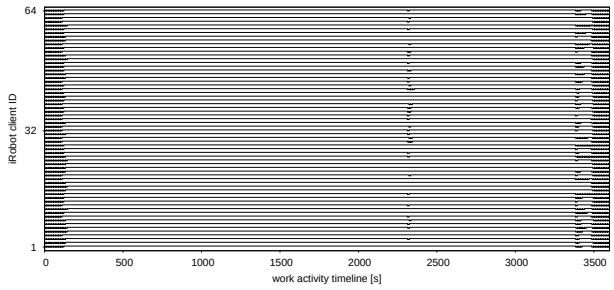

Figure 5: A test run trace obtained with 64 deployed iRobot agent instances

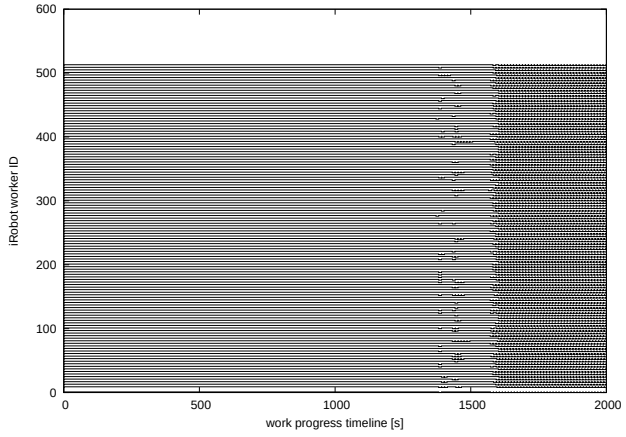

Figure 6: A test run trace obtained with 512 deployed iRobot agent instances

5 Conclusions and future work

iRobot agent proved very useful during implementation and verification of Comcute algorithms, making it possible to spot and correct some issues resulting from the 'compute as you go' approach and frequent loss of computations resulting from it. Using Python as a programming platform allowed for rapid development, easy cross platform compatibility and let designer concentrate on core agent logic and implementation of human behaviour model, instead of dealing with side tasks like access to web services or OS services.

The tool proved to be versatile, easy to reconfigure for different testing scenarios and lightweight enough to deploy large scale tests with limited resources.

Future enhancements are planned to extend the range of supported interfaces to computational platforms.

Acknowledgements: The work was partially performed within grant "Modeling efficiency, reliability and power consumption of multilevel parallel HPC systems using CPUs and GPUs" sponsored by and covered by funds from the National Science Center in Poland based on decision no DEC-2012/07/B/ST6/01516.

References

[1] SeongKee Lee, ChanGon Yoo, JungChan Park, JaeHyun Park, HaengHo Lee, WooMin Lee, and ChangMin Chung.: *Autonomous intelligent simulation model based on CGF for military training and analysis*. In Proceedings of the 10th WSEAS international conference on Computational Intelligence, Man-Machine Systems and Cybernetics, and proceedings of the 10th WSEAS international conference on Information Security and Privacy (CIMMACS'11/ISP'11), Ford Lumban Gaol and Jiri Strouhal (Eds.). World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 118-123. 2011.

[2] Seung Man Lee, Ujwala Ravinder, and James C. Johnston.: *Developing an agent model of human performance in air traffic control operations using Apex cognitive architecture*. In Proceedings of the 37th conference on Winter simulation (WSC '05). Winter Simulation Conference 979-987. 2005.

[3] Anderson, D.P.: *Boinc: A system for public-resource computing and storage.* In: Proceedings of 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA (2004)

[4] Balicki, J., Krawczyk, H., Nawarecki, E., eds.: *Grid and Volunteer Computing.* Gdansk University of Technology, Faculty of Electronics, Telecommunication and Informatics Press, Gdansk (2012) ISBN: 978-83-60779-17-0.

[5] Matuszek, M.: *Architektura Systemu Comcute.* In: Balicki J, Kuchta J., eds.: *Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid.* Gdansk University of Technology, Faculty of Electronics, Telecommunication and Informatics Press, Gdańsk (2012) ISBN: 978-83-60779-14-9. (*Comcute System Architecture*, in Polish).

[6] Czarnul P., Kuchta J., Matuszek M.: *Parallel Computations in the Volunteer based Comcute System.* In: Proc. of the 10th Intl. Conference on Parallel Processing and Applied Mathematics PPAM 2013, Warsaw, Poland

[7] Kuchta J.: *Data Partitioning and Task Management in the Clustered Server Layer of the Volunteer-based Computation System.* In: *Grid and Volunteer Computing*. Gdansk University of Technology. Gdańsk : GUT, 2012

[8] Test framework for BOINC developers, http://boinc.berkeley.edu/test.php

[9] Denaro, G., Polini, A., and Emmerich, W.: *Early performance testing of distributed software applications.* SIGSOFT Softw. Eng. Notes 29, 1 (January 2004), 94-103. DOI=10.1145/974043.974059 http://doi.acm.org/10.1145/974043.974059

[10] Kondo, D., Anderson, D.P., McLeod, J.: *Performance Evaluation of Scheduling Policies for Volunteer Computing.*

[11] Christensen, C., Aina, T., Stainforth, D.: *The Challenge of Volunteer Computing With Lengthy Climate Model Simulations.*

[12] Beberg, A.L., Ensign, D.L., Jayachandran, G., Khaliq, S., Pande, V.S.: *Foldinghome: Lessons From Eight Years of Volunteer Distrbuted Computing.*

[13] Nielsen, J.: *How long do users stay on web pages?* (2011) Nielsen Norman Group, http://www.nngroup.com/articles/how-long-do-users-stay-on-web-pages/.

[14] *Analiza możliwości rekrutacji mocy obliczeniowej w Internecie.* https://atlassian.fidointelligence.pl:8443/pages/viewpage.action?pageId=917762 (*Analysis of Computational Resources Recruitment in the Internet*, in Polish).

[15] Heien, E.M., Takata, Y., Hagiara, K., Kornafeld, A.: *PyMW – a Python Module for Desktop Grid and Volunteer Computing.* IEEE International Symposium on Parallel & Distributed Processing, 2009. IPDPS 2009.