Using XMPP for ad-hoc grid computing - an application example using parallel ant colony optimisation

Gerhard Weis and Andrew Lewis

***Abstract*—XMPP (XML Messaging and Presence Protocol), also known as Jabber, is a popular instant messaging protocol that uses XML streams for communication. Due to it's high extensibility, XMPP is very easy to adapt to other uses than instant messaging. Furthermore, announcing of presence state makes it ideal for highly volatile environments. This paper outlines the use of XMPP for a grid-like computation environment. The biggest advantage of this setup was that available computing resources, such as laboratory computers, could be connected easily and used similarly to a grid. The application example described in this paper uses Ant Colony System (ACS) optimisation and the NEC-tool to optimise RFID antennas, involving computing the efficiency and resonant frequency of a large number of different antenna structures.**

I. Introduction

XMPP [9], [10] forms a peer to peer (P2P) network, and is widely used for instant messaging. The architecture of XMPP requires that each peer connects to a server and each server hosts a domain. XMPP endpoint addresses are very similar to email addresses with the addition of a node name. The address *user@example.com/nodename* describes a user with the name *user* known at the domain *example.com* and the instance with which the user is connected to the server is called *nodename*. XMPP servers of different domains can be interconnected so that messages from peers from one domain are routed to users from different domains connected to other servers. By adding so called "gateways" to servers, it is even possible to connect XMPP networks with other P2P networks. Such gateways log into the foreign network on behalf of the user and translate all messages between the two, different protocols.

An XMPP server is basically an XML router that routes small XML-snippets to the addressed endpoint. As XMPP is optimised to route small XML-snippets (stanzas) from one peer to another peer, transmitting large data blocks has to be done by other communication channels like ftp, http, etc. However, two peers can easily negotiate a suitable out-of-band communication channel.

Extensibility in XMPP is achieved by using XML-namespaces. Each 'sub-protocol' defines it's own XML namespace, and hopefully some document with a detailed description of the purpose of the 'sub-protocol' and the XML elements allowed for this namespace.

XMPP offers two message transmission patterns. A 'Request/Response' mechanism and a 'Request/maybe Response' mechanism. The latter one is used for instant messaging, where message not necessarily require an answer, or an answer can be unrelated. However, this communication pattern in XMPP allows tracking of related messages. The 'Request/Response' mechanism is mostly used for XMPP and P2P management tasks.

Gerhard Weis completed his MICT degree with Griffith University (email: gerhard.weis@gmx.com). Andrew Lewis is with the Institute for Integrated and Intelligent Systems, Griffith University, Queensland, Australia (email: a.lewis@griffith.edu.au).

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

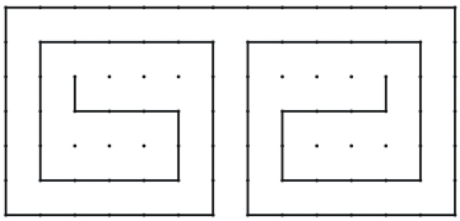

Fig. 1. Example structure for a meander line RFID antenna

II. The example application

The work described in this paper arose from the practical necessities inherent in performing computational optimisation studies of meander line RFID (radio frequency identification) antennas. Meander line antennas are a subset of particular interest for RFID as, by their nature, they are compact and tags may be readily manufactured. These antennas normally have a planar structure and consist of printed conductive tracks on thin plastic substrates [4]. Half the antenna is modelled and the half-antenna element mirrored to form a dipole antenna. Figure 1 shows one possible structure of the meander line antenna.

The automated design of these antennas is relatively new. Traditionally, design engineers would create these manually in a time consuming iterative process, as there is no generally applicable analytic design methodology for meander line antennas. However, this was only practicable for small and relatively impractical antennas. Recent work has shown that heuristic search techniques are capable of producing very good antenna designs for larger problems [8], [12]. One approach to design of the general meander line antenna is to confine the antenna to a Cartesian grid. For a particular grid defining the meander line, a number of design choices are possible. This gives rise to a combinatorial optimisation problem similar to the travelling salesman problem (TSP), and as such, can be solved by meta-heuristic search strategies. In the work described in this paper an ant colony optimisation algorithm, adapted for multi-objective optimisation by using a Pareto-dominance relationship to determine solution fitness [13], has been used.

When it comes to the practical implementation of the approach, the problem is that, for the ant colony system (ACS) algorithm to create a RFID antenna structure, many different solutions have to be evaluated for efficiency and resonant frequency. Each antenna simulation is rather CPU intensive but, as the evaluation of the performance of each antenna is independent from any other, it is possible to perform the antenna evaluation in parallel to reduce the overall time taken. In addition, this problem is attractive for a distributed computational framework because the description of an antenna structure to the simulation evaluating its performance, and the return of the resulting performance data, requires the transfer of very little data, minimising communication overheads.

In the work reported in Weis *et al.* [12] ACS with local

search heuristics was used to optimise the gain of meander line RFID antennas. The ACS algorithm constructed a trial antenna configuration by constructing the path of an antenna element on a Cartesian grid of points. Antenna performance was then evaluated using the NEC tool [2], an open source antenna simulation program that provides a library of functions to analyse antennas. It takes an input file that describes the antenna structure and various other simulation parameters and executes a simulation. As the only values of interest in the study in question were efficiency at the resonant frequency, the NEC main program was arranged to perform a binary search for the resonant frequency within a frequency band of interest. At its conclusion the simulation returned values of resonant frequency and the corresponding efficiency to the ACS program, which proceeded with further iterations.

The ACS algorithm was permitted 2000 iterations with a population of 10 ants for each of 10 random, initialisation seeds for each antenna grid size. The local search heuristic was allowed a variable-depth tree search. Thus, at any single iteration there might be anything from 300 to 3000 simultaneous requests for antenna evaluations. To reduce the computational cost a persistent cache of antenna configurations and corresponding performance was maintained and inspected before passing the request to the NEC program. Nevertheless, several thousand evaluations might be required for each antenna grid size.

III. The computational resources

The student computer labs at Griffith University are equipped with desktop computers that are available for use throughout the day, and sometimes also at night, but often are not used the whole time. These "free" computing resources are ideal for use for an *ad-hoc* grid computing solution. The system described in this paper was tested on approximately 30 Pentium 4 CPU-based computers and 16 Athlon X64 dual core CPU-based computers. As 16 of the Pentium 4 computers were located in an open user room, many of them were occasionally turned off or rebooted into Microsoft Windows™ during the test runs, temporarily precluding their use for the Linux-based simulations. The main controller, on which the XMPP server and the ACS algorithm ran, was a very old, 400MHz Pentium 3 CPU-based computer with 128MB of RAM. It ran NetBSD 4.0, with jabberd as the XMPP server. The ACS algorithm and the XMPP monitor were implemented in Python 2.5. The worker computers mount a shared "home" directory via NFS which was used to exchange data between nodes. In particular, the NEC tool and the worker-XMPP program were installed in this NFS filesystem. Two versions of the NEC tool were compiled; one optimised for the P4 machines and one for the Athlon-X64s. All computers allowed ssh-login with a user id stored in a central, LDAP directory.

As the lab computers are also used for classes, it was desirable that the NEC tool should not run when another student was using one of those computers. For this reason the worker program detected any user login, and stopped accepting computation tasks until the computer was once more vacated. In XMPP such a node announces itself with a presence state of 'Do not disturb' (DnD). If a computation request arrives during a DnD-state, the node returns an error message to let the master know that the request has to be rescheduled.

IV. Implementation

As the ACS algorithm has a rather short run time compared to the antenna evaluations, it is not a performance critical part. For this reason it was decided that Python could be used as the implementation language, because it is well known for it's rapid prototyping qualities (see, for example, Chaves *et al.* [3].) The ACS algorithm was also ported to Python, to make it easier to manipulate various run time options. For the XMPP communication layer, the simplest, readily available Python XMPP library, 'jabber.py' [1], was used. The persistent antenna cache used a Berkeley DB [7], [6], to allow more memory-efficient storage of the large amounts of data accumulated.

The controller (job scheduler) is a simple XMPP-peer. It runs the ACO algorithm to generate the RFID antenna structures and sends the generated structure descriptions to the available computation nodes. The scheduler implemented uses only presence information from the computing nodes and does simple round-robin scheduling.

The controller node contains a static list of available computation nodes and the main program is split into three threads. One thread continually scans through the list of offline nodes and if an ssh connection is possible, this thread tries to start the worker peer program as a user daemon. The second thread runs the ACS algorithm. This thread places generated antenna structures in a synchronised queue. The main thread handles all of the XMPP-related messaging and is also the main loop. The main loop uses the 'select' system call to wait for arriving XMPP messages. As this blocks the whole main thread, a timeout of 1 second is used to break out of this waiting state if no incoming XMPP messages are available, and send out further jobs if possible. On arrival of presence messages, this thread updates the internal list of available worker nodes accordingly. If a new structure to evaluate is in the queue, it will be sent to a worker. The main program also keeps track of which structure is sent to which node because, in case of errors or if a node goes offline (fails) without returning a result, the antenna structure has to be resent to another node for evaluation. If a result arrives from a worker node it is placed into a persistent database. The ACS algorithm can also access this database, so that antenna structures already calculated do not have to be recalculated. The main loop also offers a chat bot interface. This interface offers a command line-like user interface, with which the controller program can be monitored and managed. A similar chat interface is also available for the worker programs. The control flow of the job scheduler is shown in Figure 2.

To transmit a job to a worker a small 'sub-protocol' was designed for the 'Request/Response' mechanism offered by XMPP. The top level element for this was an *iq* element. An *iq* element has the destination address, a type (get, set, error or result) that defines the semantics of this message, and a unique id which allows relaying a possible answer to the request. Some namespace attributes are stored ss child elements in this *iq* element. XMPP also requires a *query* element. This *query* element carries information about the actual payload sent with the "iq"-request, such as XML-namespace. The actual payload can be found under the *query*.

The following shows an example request for an 8x8 antenna.

```
<iq to='work1@x.y.z/worker' type='set' id='5'>
  <query xmlns = 'my:iq:exec' >
    <vector size='8'>8 7 15 16 24 23 31 39 38
30 22 14 6 5 4 3 2 1 9 17 25 33 41 49 57 58 59
60 52 51 50 42 34 26 18 10 11 12 13 21 20 19 27
28 29 37 36 35 43 44 45 53 61 62 54 46 47 55 63
64 56 48 40 32</vector>
  </query>
</iq>
```

When a node returns a result, it sends back an 'iq'-message of type 'result' or 'error'. The following shows an example response. It was decided to include all the task information in the response because the payload itself is very small and it allows the monitor to double check results.

```
<iq from='sentret@x.y.z/worker' type='result'
id='28' to='monitor@a.b.c/monitor'>
```

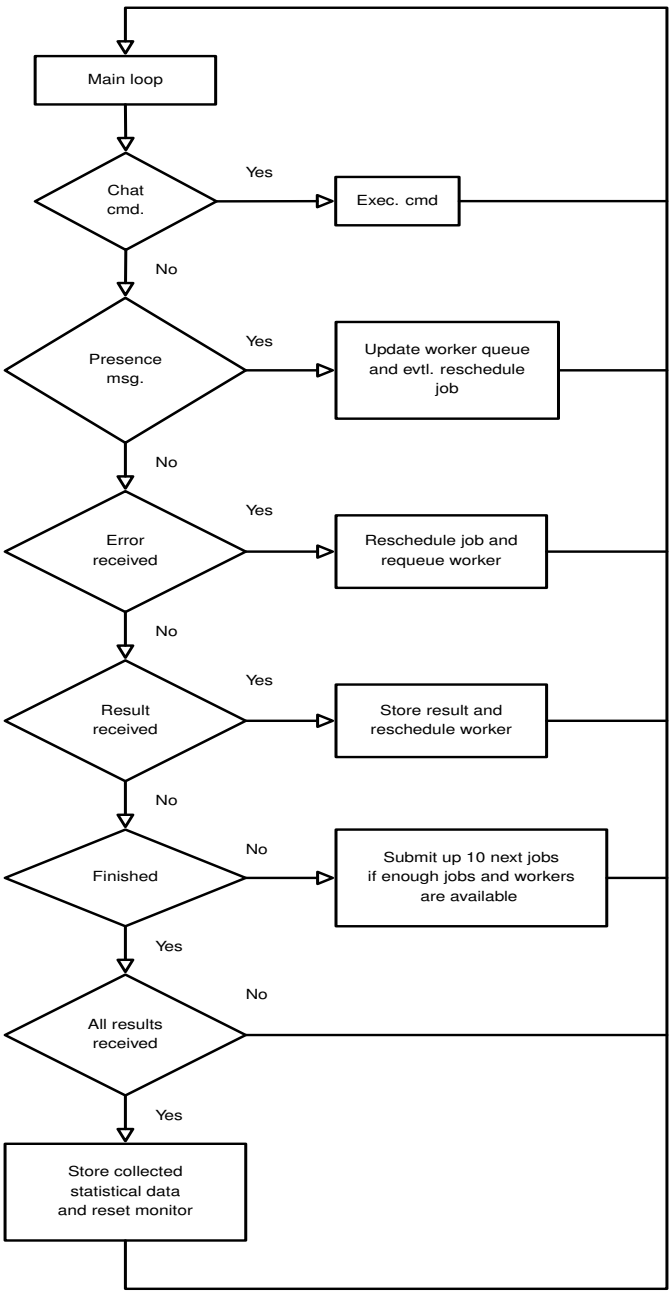

Fig. 2. Flow diagram of job scheduler

```
  <query xmlns = 'my:iq:exec' >
    <vector freq="750" eff="750" size='8'>8 7
15 16 24 32 40 48 56 64 63 55 47 46 54 62 61 53
45 37 29 28 36 35 27 19 20 21 13 12 11 10 18 26
34 42 50 51 43 44 52 60 59 58 57 49 41 33 25 17
9 1 2 3 4 5 6 14 22 30 38 39 31 23</vector>
  </query>
</iq>
```

For this project a complete, new (very simple) 'sub-protocol' was implemented to meet the needs of the experiment. However, there are also generic discovery and command execution 'sub-protocols' defined for XMPP. These protocols tend to be rather complex, but may be a good solution for more generic tasks.

V. Results

This very simple setup and program provided a useful platform for the computational experiments required for the antenna design application. A number of antenna designs, for a number of grid densities, were optimised using ACS and then further refined using local search, The 5×5 was quite impressive. Beginning with a search depth of 5 it took only 3 steps to find the global optimum as determined by Galehdar et al. [5] using exhaustive enumeration. The total number of antennas evaluated was 13. For the 6×6 grid local search depth was set to 25. Two near-optimal candidate antenna structures were generated by the ACS algorithm using different operating parameters. For the first antenna, the local search algorithm needed to evaluate 270 antennas and improved the antenna efficiency by 1.5%. For the second structure, there were 395 antennas to evaluate. The efficiency was improved to by 21.9%, eventually reaching the same efficiency as the first. It might be inferred from this that these searches had both reached near-optimal results for this antenna size, but that the first candidate had a better result originally.

For the 7×7 case local refinement produced 891 antennas to evaluate (for a search depth of 25) and found 7 different equally efficient antenna structures, with an increase in efficiency of 7.8%. To reduce computation time the search depth for 8×8 grids was reduced to 20. The local search had to evaluate 3193 different antennas and improved the efficiency 7.6%. For the 9×9 grids, the search depth was further reduced to 15, because larger grids take increasing compute time to evaluate. For this search depth 1371 antennas had to be evaluated. A solution was found with efficiency increased by 16.8% compared to the original structure. For 10×10 grid, the search depth was also reduced to 15 and 1527 new antennas were evaluated. The efficiency was improved notably, by 42.7%. Comparing the resulting antennas to the "state-of-the-art", it is known that for a simpler, single-objective problem, the approach used was able to find the globally optimal structure on a 5×5 grid, by reference to results from experiments that exhaustively enumerated all possible solutions [12].

As the computation time for one antenna structure is rather long, and the data to describe a job is very short, this processing framework scaled very well.

The data in Table 1 shows the timing data of a recalculation run. During such a recalculation run, all known antenna structures are sent out to the computation nodes, to recalculate their frequency and efficiency properties. This means that no optimisation was involved during this calculation. This particular scenario gives a good indication of how well the framework presented works. Table 1shows specific data for each grid size. The value for 'X64 nodes' and 'P4 nodes' indicate how many of each of those computers have been used, 'structures' is the number of antennas calculated. The row 'est. X64' shows the theoretical time it would have taken to simulate all antennas on one X64 (dual core) node. The time is based on the average time it took to simulate one antenna, for the specific grid size, on this type of computer. The 'wall clock' row is the time it took to simulate all structures with the program presented here, and 'speed up' is the factor between 'wall clock' and 'est. X64'. The data in 'overhead' is a rough estimation of how much time has been wasted by the monitor program and XMPP protocol. It compares the maximum time spent purely for antenna simulation during the whole experiment to the wall clock time.

Figure 3 graphs "speed up" and "overhead" for each grid size. While a significant speed up was achieved over all grid sizes, it is clear that the longer the simulation of one antenna structure takes, the greater the improvement and the less the overhead. Unfortunately, it is not clear whether it was the

grid size	5	6	7
X64 nodes	7	7	7
P4 nodes	1	2	4
structures	13236	35888	65027
est. X64	53:48	3:12:08	7:17:20
wall clock	16:08	43:24	1:32:55
speed up	3.34	4.43	4.71
overhead	46.23%	35.33%	31.84%
grid size	8	9	10
X64 nodes	7	7	7
P4 nodes	6	5	5
structures	103262	122373	129791
est. X64	13:37:03	21:40:00	39:05:40
wall clock	2:47:04	4:07:59	5:26:08
speed up	4.89	5.24	7.19
overhead	24.96%	19.44%	15.68%

TABLE I
TIMING DATA.

XMPP protocol itself or just an overload of the monitor node (the old P3) that caused such relatively poor improvement for smaller grid sizes compared to larger grid sizes. Another possible reason is that the monitor program is optimised for rapid development instead of run-time performance, which leads to a rather awkward implementation of the inner threading and queuing. Results quoted in other studies [11] suggest that it would be reasonable to expect the server performance could be scaled by one or two orders of magnitude. Since there are sufficient antenna evaluations presented at each iteration, each of which is completely independent of all others, to scale to a similar degree, it may be possible the application has significant potential to scale to problems with larger grid sizes. These problems could also be expected to have improved ratios of computation to overhead, and thus improved speedup as suggested by the trend in Figure 3. At some point, however, there will be an absolute limit on the ability of a particular server to handle the volume of messages. This perhaps could be addressed through a replicated server architecture.

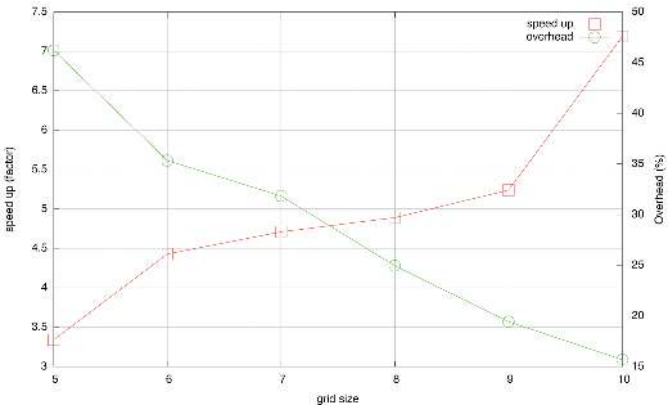

Fig. 3. Speed up and overhead over grid size.

VI. CONCLUSION

This paper describes an *ad hoc* grid framework implemented to provide processing resources to a practical use of ant colony optimisation. The advantages of this approach were the minimal effort to build the system and the ease of deployment, since no special access rights were required for setup. Additionally, all programs were built with tools that are available on almost any standard Linux system installation and the whole installation footprint is very small. Although the system was built from scratch, the development time was very short and it is still easily extensible. Furthermore, the reliance on standard components provided overall system stability.

Despite various problems with system load and awkward implementation, this setup has been shown to be efficient enough to speed up a real-world optimisation application by a factor of nearly 8 using only a moderate number of desktop computers. More work is needed to investigate some of the implementation issues in more detail, to determine if further improvement is possible. To this end, attempts will be made to optimise the monitor program and try to run it on a more powerful machine. In addition, better measurements of the protocol overhead are planned and further developments will be made to try to achieve a greater utilisation of the worker nodes. A more sophisticated scheduler that, for instance, prioritises nodes on the basis of their demonstrated computation time might also further reduce overall simulation time.

REFERENCES

[1] M. Allum. jabber.py – A Python Jabber library, Oct 2004. http://jabberpy.sourceforge.net/, Last checked 12.12.2008.

[2] G. Burke, A. Poggio, J. Logan, and J. Rockway. NEC - Numerical electromagnetics code for antennas and scattering. *Antennas and Propagation Society International Symposium, 1979*, 17:147–150, June 1979.

[3] J.C. Chaves, J. Nehrbass, B. Guilfoos, J. Gardiner, S. Ahalt, A. Krishnamurthy, J. Unpingco, A. Chalker, A. Warnock, and S. Samsi. Octave and python: High-level scripting languages productivity and performance evaluation. In *HPCMP Users Group Conference, 2006*, pages 429–434, 2006.

[4] K. Finkenzeller. *RFID Handbook: fundamentals and applications in contactless smart cards and identification*. Wiley, Chichester, England, 2nd edition, 2003.

[5] A. Galehdar, D. Thiel, and S. O'Keefe. Antenna efficiency calculations for electrically small, RFID antennas. *IEEE Antenna and Wireless Propagation (in press)*, 2007.

[6] M.A. Olson, K. Bostic, and M. Seltzer. Berkeley DB. In *ATEC '99: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 43–43, Berkeley, CA, USA, 1999. USENIX Association.

[7] Oracle Berkeley DB. http://freshmeat.net/redir/berkeleydb/694/url_homepage/berkeley-db/, Last checked 12.12.2008.

[8] M. Randall, A. Lewis, A. Galehdar, and D. Thiel. Using ant colony optimisation to improve the efficiency of small meander line RFID antennas. In 3rd *IEEE International e-Science and Grid Computing Conference*, pages 345–351, Washington, DC, USA, 2007. IEEE Computer Society.

[9] P. Saint-Andre. Extensible messaging and presence protocol (XMPP): Core. RFC 3920 (Proposed Standard), October 2004.

[10] P. Saint-Andre. Extensible messaging and presence protocol (XMPP): Instant messaging and presence. RFC 3921 (Proposed Standard), October 2004.

[11] S. Schulz, W. Blochinger, and M. Poths. A network substrate for peer-to-peer grid computing beyond embarrassingly parallel applications. In *International Conference on Communications and Mobile Computing (CMC 2009)*, 2008, accepted.

[12] G. Weis, A. Lewis, M. Randall, A. Galehdar, and D. Thiel. Local search for ant colony system to improve the efficiency of small meander line RFID antennas. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2008)*, 2008.

[13] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.