Lecture Notes in Computer Science
Pages 531-538, Vol 3320, 2004

Tuning Genetic Algorithms for Real Time Systems using a Grid

A. Martí Campoy, F. Rodríguez, A. Perles

Departamento de Informática de Sistemas y Computadores
Universidad Politécnica de Valencia
Camino de Vera s/n 46022 SPAIN
amarti@disca.upv.es, prodrig@disca.upv.es, aperles@disca.upv.es

Abstract. The use of locking caches has been recently proposed to ease the analysis of the performance and predictability of a cache when used in a real-time system. One promising method to adequately select the cache contents is the use of a genetic algorithm. However, this method requires the tuning of analysis parameters and this step requires a huge computational cost that can be reduced only if a massively parallel computing infrastructure is used. The work presented here analyses the specific requirements of the genetic algorithm tuning and the facilities provided by commercial grid software. Although the grid eases the resource management and job execution it lacks some communication link with submitted jobs, which is solved by the use of a specialized program called the Experiment Manager. This experiment manager supplements the grid and offers a completely automated environment for algorithm tuning to the researcher.

1 Introduction

Locking cache has shown its goodness as an alternative to dynamic, conventional caches for its use in real-time systems. The use of a locking cache offers two advantages: its behavior is fully predictable, allowing the system designer to compute the response time of tasks in a simple way and it provides similar performance than conventional, non-predictable caches [1] [2] [3]. The operation of a locking cache is simple: a sub-set of instructions, belonging to any task in the system, is loaded and locked in cache during the system start-up. Those instructions remain in cache for the whole system operation; that is, there are no cache replacements. In this way, the behavior of the cache is fully predictable and execution times can be easily estimated.

The resulting performance is highly dependent on the loaded and locked instructions. The response time of tasks, their slack time, and global system utilization varies as a function of what instructions are locked in cache. Adequate selection of these instructions is a hard problem because the number of possible solutions is huge. Equation 1 shows the number of solutions, where *n* is the number of main memory blocks occupied by all the tasks in the system, and *m* is the number of lines of the locking cache. As an example, a system of 500 main memory blocks (around 8Kbytes) and a locking cache of 250 lines (around 4Kbytes), requires the analysis of more than 10^{149} possible solutions.

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} \qquad (1)$$

In this kind of problems, genetic algorithms [4] are a very interesting solution. In [5] a genetic algorithm is presented, that efficiently solves the problem of selecting the locking cache contents. This algorithm was developed in the C language, and its execution time in a medium performance computer is between four and twelve hours, depending on the experiment parameters. Although this time may appear too large, the cache content selection is accomplished once during system design, so this computational cost is relative.

2 The problem

Results obtained from the genetic algorithm are, in general terms, good. But these results vary widely with several parameters, both algorithm parameters – seed, population size, number of generations, probabilities of mutation, crossover, etc…- and system parameters – cache size, particular set of tasks, cache mapping function, etc…-. Table 1 shows the main parameters that may affect the goodness of

algorithm results with their range of possible values. Value range of parameters is established from typical values found in literature, or preliminary experimentation.

Genetic algorithm runs using different values for these parameters must be carried out to identify the optimal configuration of the genetic algorithm, providing: i) the best performance when locking cache is used, or ii) a relationship between system and algorithm parameters in order to best configure the algorithm for each particular system configuration.

The first issue in this work is the computational time needed to accomplish all the experiments. Considering an average execution time of 6 hours for the algorithm, more than 11,000 years are needed to perform them using a single computer. But the processor time is not the only temporal issue. Setting, loading, launching and processing the experiments requires a huge human researcher time. Next section presents how to minimize these temporal costs.

Another important issue is the number of replicas with different seeds. Since genetic algorithms perform a pseudo-random search, the effect of the initial seed may be significant. But the number of replicas may vary for each experiment in order to achieve the required confidence level on the results, so it is not easy to determine a-priori how many replicas are required for each configuration.

Table 1. Experimental values of parameters

Parameter	Value
Seed	Between 10 and 100 different seeds
Population size	100,150, 200, 250
Number of generations	From 1000 to 10000 in 10 steps
Selection probability	0.05, 0,1, 0.2
Mutation probability	0.001, 0.002, 0.003
Crossover probability	0.4, 0.5, 0.6
Cache configurations	50 different cache configurations
Set of tasks	30 different sets, synthetically created

3 The proposal

To make possible the experiments, several requirements must be accomplished:

- The number of cache configurations and set of tasks must be reduced. This is possible grouping systems and using a factorial design of experiments [6].
- Replication of experiments with different initial seeds must be dynamically determined. That is, confidence interval at desired percentage of confidence level for average result must be computed at run time and new replicas will run if needed only.
- Effect of number of generations must be obtained as partial results, setting the maximum number of generation in a dynamic way, and not as static, pre-defined value.
- A massively parallel computing infrastructure must be used, like grid computing [7].
- Automatic setup and run-time post-process of the experiment results is needed.

Using factorial design of experiments the number of required experiments is drastically reduced because not all the possible combination of parameters must be analyzed (this does not apply to the initial seed and number of generations, their effect must be computed as results are available). This reduces the time needed to perform them to around three years of a single computer. Using grid-computing techniques this time may be also reduced.

Innergrid [8] is a grid software that allows to use, in an easy way, interconnected, heterogeneous computers to distribute jobs. Innergrid architecture is based in one server and several agents. Agents run in each of the networked computers, and are inquired by the server to run the programs demanded by the grid's user. The grid's user submits the set of jobs to be executed to the server and the server polls the agents to find non-busy computers to run those jobs. Also, Innergrid takes care of questions like security, fault tolerance and system scalability making its use very simple. Fig. 1 depicts the Innergrid architecture and operation.

Innergrid is used today by several research groups in the Technical University of Valencia as isolated grids of a dozen computers each, testing the environment before around 3000 computers are interconnected in one large, high performance grid, integrating computers devoted to research and teaching purposes. Although the usefulness of Innergrid for the here presented problem is evident, there are two operational characteristics that preclude the direct use of Innergrid:

- The server offers a web-based interface to submit the jobs that must be run in the grid. This interface is very human friendly but there is no provision to ease automation of this interface by means of scripts or similar. Also, this web-based interface is the only way to retrieve results from the jobs once they finish their execution.
- There is no facility to provide a communication link with the jobs running in the agents. This way, run-time computing the confidence interval for results and dynamically determining when there are enough runs of one experiment is not possible.

4 Experiment automation: The Experiment Manager

This section describes the work developed to automate the experiment setup and to provide a communication link with submitted jobs. This capabilities are carried out by a specialized program called the Experiment Manager (EM). This experiment manager may run in any computer that can connect to the grid server, and performs three main operations:

1. Connects with the Innergrid server and load the set of experiments into the grid. The set of experiments is read from a configuration file, created by the human researcher, where the factorial design is defined. The desired confidence interval for the results and the threshold to determine that no new generations are needed is also specified into this file. In order to connect with the Innergrid server, the EM implements a background http [9] client without graphic interface but capable to interpret the html documents received from the server and upload the forms.
2. Communicates with genetic algorithms running in all agents, receiving partial results. During the execution of those algorithms, the EM computes the confidence interval. If the error from the confidence interval is not as small as specified in the EM configuration file it connects again with the Innergrid server and launches new replicas into the grid. Also, the improvement of the function fitness between generations is computed. If this improvement overpass the threshold, the EM connects with the server to finish the jobs.
3. When a set of experiments finish execution in the grid, the EM connects with the Innergrid server and download results, storing them in a tabular format to allow an easy and fast processing by statistical tools.

Fig. 1 shows the Innergrid architecture supplemented with the EM, and the relationships between the different elements.

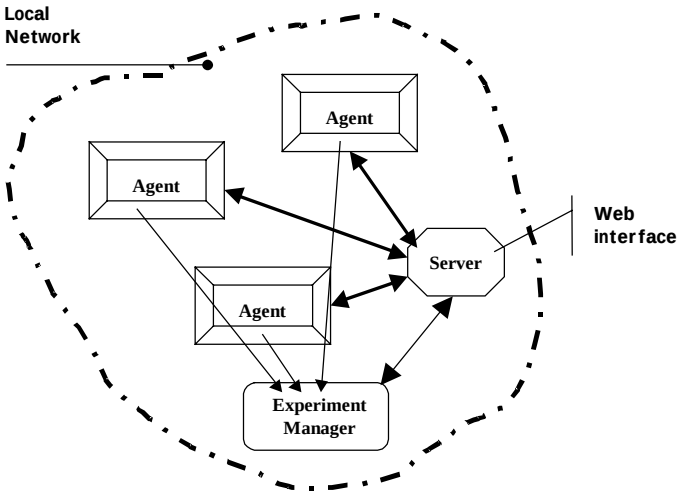

Fig. 1. Innergrid scheme with EM and relationship between elements

5 Implementation and operation details

This section describes some important remarks in the implementation and use of the several elements involved in the described experimental process

Genetic algorithm

The genetic algorithm has suffered several changes from the first version, written in the C language. In order to allow its execution in several architectures with different operating systems the algorithm was rewritten using the Java language. And inter-process communication links have been added using sockets [9] and TCP/IP. Several operating parameters – like finish condition, provided output, initial seed and system configuration – are loaded from files.

Experiment Manager

Five modules form the experiment manager: Configuration parser, client, communications, analyzer, and database.

The configuration parser reads the configuration file and creates the particular files needed by the Innergrid server. The EM http client sends these files to the Innergrid server. This http client can process a reduced set of html tags, only those used by the Innergrid server. For each entry in the factorial design, a grid experiment –formed by several replicas– is created.

The communication module accepts connections from each instance of the algorithm running in the grid and receives their results. With these results, the analyser determines if new replicas are needed or if the algorithm may finish its execution in the current generation. If new replicas are needed, the client submits them to the grid as new experiments. The amount of data the EM must manage is very large, because confidence intervals and threshold of improvement is computed for isolated subsets of experiments. The database module offers the functionality required to manage all the data efficiently.

Fig. 2 shows how the modules interaction.

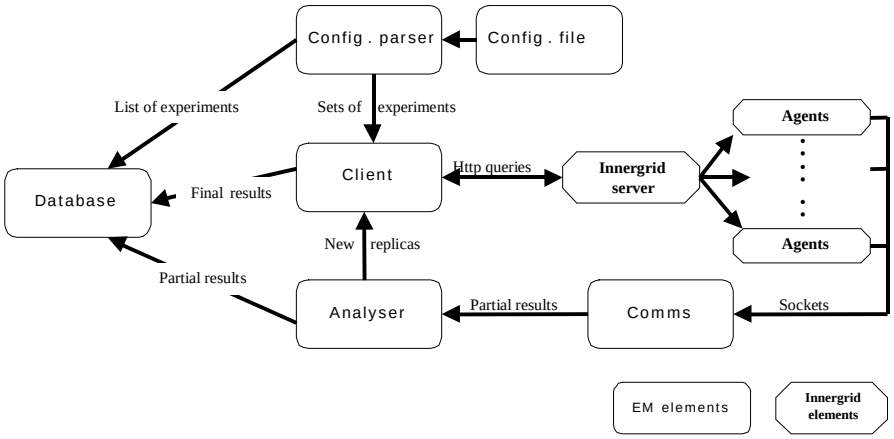

Fig. 2. EM modules and their interactions with Innergrid

Experiment framework

The only work the researcher must do to run the experiments is creating the configuration file for the EM. This file includes the factorial design that may be easily created using statistical software like Statgraphics [10], and other parameters like confidence interval, improvement threshold, initial number of replicas, maximum number of generations, and step of generations to produce partial results.

It is clear that parameters in the factorial design – number of systems, range of different probability values, number of cache configurations – indirectly determines the amount of time needed to accomplish the in-depth evaluation of the generic algorithm, because these parameters determine the number of experiments that must be ran. But other parameters have a direct impact on the experimental time because the directly affect the throughput of the grid:

- Initial number of replicas: naively the number of initial replications may be set to a large value [11]. This cause, for some experiments, the execution of non-necessary instances of the experiment delaying the execution of other experiments while grid resources are busy doing nothing significant. Initial number of replicas must be set to a value that allows the grid to run simultaneously different entries of the factorial design, taking advantage of the EM ability to dynamically compute and launch the replicas as they are needed.

- Step of generations: when the genetic algorithm reaches a step in the number of generations, connects with the EM and sends the results. Setting a low value for this parameter results in an increase of network traffic. Taking into account that the grid shares the network with processes of other users, the traffic generated must be restrained.

6 Conclusions

The use of locking caches has been recently proposed to ease the analysis of real-time systems that use a cache to increase its performance. One promising method to adequately select the cache contents is the use of a genetic algorithm. However, an in-depth evaluation of the genetic algorithm is required to tune its parameters and this process needs huge computational resources, even when a factorial design of the experiments is used.

The "griddization" process is not easy when the number of execution runs or the algorithm finish condition may be not statically, a-priori defined. This is the case of experiment setups where the confidence interval for each result must be obtained with multiple executions of the same program using different seeds, for example.

The grid used to carry out the experiments offers an abstraction to the user of a large computer system where many independent programs may be submitted, performing a complex resource management policy to map jobs to lightly loaded computers from a heterogeneous network.

However, the grid does not provide an adequate abstraction to execute a large number of jobs with inter-process communication in which executing programs may submit partial solutions or intermediate results to a central repository. Submitted jobs are assumed to be independent and the results are available once the program has finished only. This lack of flexibility has led the authors to supplement the grid with specialized software, the Experiment Manager (EM), to provide the programs executing in the grid this result repository that may be accessed at any time using TCP/IP sockets.

This experiment manager complements the services offered by the grid and provides a completely automated environment for the genetic algorithm tuning process or any other situation where the grid abstraction is not enough and must be supplemented with inter-process communications.

The EM, coupled with the resource management provided by the grid, offers the possibility to the researcher to dynamically increase the number of submitted jobs to achieve the desired confidence interval. The EM may be also used to cancel already submitted jobs to minimize the number of replicas executed when the confidence interval has been reached increasing the grid utilization.

References

1. Martí Campoy, A., Perles, A. Busquets-Mataix, J.V.: Static Use Of Locking Caches In Multitask, Preemptive Real-Time Systems. Proceedings of the IEEE Real-Time Embedded System Workshop. London, UK, December 2001
2. Puaut, I. Cache Analysis vs Static Cache Locking for Schedulability Analysis in Multitasking Real-Time Systems.: Proc. of the 2nd International Workshop on worst-case execution time analysis, in conjunction with the 14th Euromicro Conference on Real-Time Systems, Vienna, Austria, June 2002
3. Vera, X., Lisper, B., Xue, J.: Data Cache Locking for Higher Program Predictability. International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), pages 272--282 San Diego, California, June 2003. ACM Press
4. Mitchell, M.: An Introduction to Genetic Algorithms. Mit Press. 1996
5. Martí Campoy, A., Pérez Jiménez, A., Perles Ivars, A., Busquets-Mataix, J.V.: Using Genetic Algorithms in Content Selection for Locking-Caches. Proceedings of the IASTED International Symposia Applied Informatics. pp. 271-276. Acta Press. Innsbruck, Austria February 2001
6. Jain, R.: The Art of Computer Systems Performance Analysis. Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley- Interscience, New York (1991)
7. Foster, I., Kesselman, C.: The Grid. Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, USA. 1999.
8. InnerGrid, by GridSystems. http://www.gridsystems.com
9. Kurose, J.F., Ross, K.W. : Computer Networking. A Top-Down Approach Featuring the Internet. 2nd edn. Addison-Wesley (2003)
10. Statgraphics plus. http://www.statgraphics.com
11. Law A. M., Kelton W. D.: Simulation Modeling and Analysis. McGraw-Hill. (2000)