AN IMPROVED EVOLUTIONARY APPROACH FOR THE CLUSTER ALLOCATION PROBLEM

APOLLONI R, MOLINA S., GALLARD R.H.

Proyecto UNSL-338403[1]
Departamento de Informática
Universidad Nacional de San Luis (UNSL)
Ejército de los Andes 950 - Local 106
5700 - San Luis, Argentina.
E-mail: {rubenga, smolina,rgallard}@unsl.edu.ar
Phone: + 54 652 20823
Fax : +54 652 30224

Abstract

In a distributed system, consisting of a set of interconnected local area networks, users migrate to different machines, users invoke different programs and users and programs need distinct data files to satisfy their expectations. Consequently optimal allocation of parallel program tasks can increase system performance as results of traffic cost reduction between clusters[2].
The problem of allocating a program in a particular system node can be divided into two subproblems: i) allocate the program in a cluster such that traffic costs are minimized and ii) within a particular cluster choose the node following some load balancing criteria [5].
To solve subproblem i), in 1992 U. M. Borghoff [2] proposed the Individual Program Execution Location Algorithm IPELA, where essentially giving a distribution of data files the best allocation for program execution, minimizing the expected intercluster traffic, is searched.
The algorithm uses diverse input data such us the cost for starting a program at some node [10], the dependencies between program and data files [1], separated read and write access costs [9], the impact of I/O activities on the communication costs [8] and the allocation of program and data files [3].
As the number of possible allocations induce high complexity and the model could not be solved to optimality Borghoff reduced the number of combinations by limiting the number of data file replicas and looking for those combinations where the relevant file sets's allocation is varied. This approach reduced complexity. Nevertheless running IPELA implied evaluation of each solution in a large problem space.

Extending the Borghoff's individual program framework, we focussed on its application in a parallel program environment confronting the problem by means of an evolutionary approach [6].

---

[1] The Research Group is supported by the Universidad Nacional de San Luis and the ANPCYT (National Agency to Promote Science and Technology).

[2] Nodes belonging to the same local area network are said to build a cluster.

The current presentation discusses possible improvements of the original evolutionary approach at the light of enhancements found in the evolutionary field itself.
Consequently two new recombination methods: multiple crossovers per couple (MCPC) and multiple crossover per mating of multiple parents (MCMP) will be applied to the evolutionary algorithm for the cluster allocation problem. Details on experiments and results original approach are shown.

2. The model

Now we briefly explain the nature of the problem. Given a user initiated parallel program, which during execution accesses to a set of files in a distributed system, allocate the program parallel tasks (modules) in order to minimize intercluster traffic (tasks migration, intermodule and module-file communication).

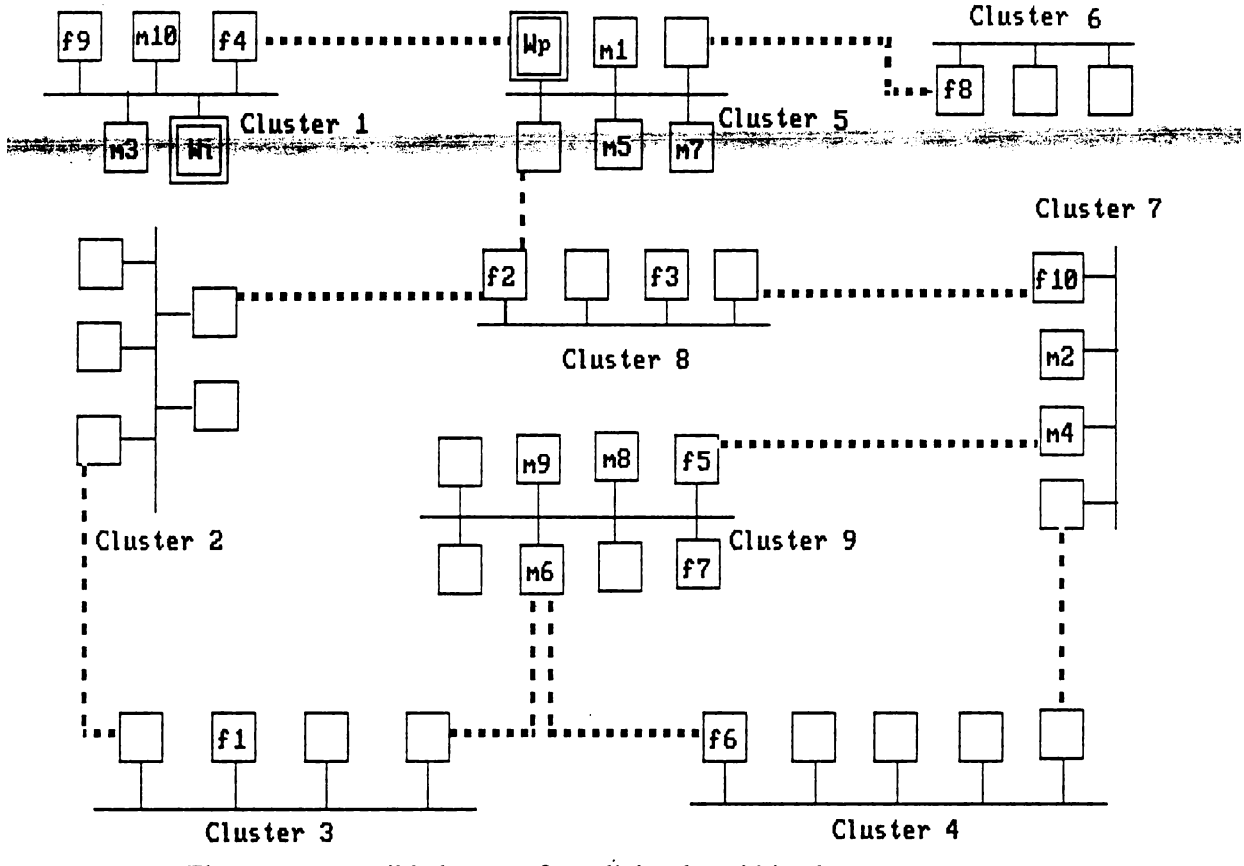

Figure 1. A possible layout of parallel tasks within clusters.

As an example Fig. 1 shows the final allocation of parallel tasks after both substrategies were applied. The parallel program, comprising ten tasks, was residing in workstation Wp of cluster 5, initiation took place from workstation Wi of cluster 1, ten files were distributed throughout the system and the parallel tasks, were first allocated to nine available clusters following the minimization criteria and then to a selected node, within their respective clusters.

In other words, our main objective is to minimize communication time between the clusters where the parallel program components are allocated under the following characteristics:

- The program, which is residing in some cluster, is decomposed in M parallel modules where each of them can be executed in an arbitrary available[3] cluster.
- During execution, parallel modules can communicate each other and access data blocks in files which are resident anywhere in the system.

In our model some data structures referring to programs profile and file distribution are supposed to be available in a Cluster Supervisor Node (CSN) within each cluster.

Assuming that,

- A parallel program can be divided into M migrating modules.
- Each migrated module generates a number of output data blocks (transient or partial results) to the node where it is residing. These blocks are to be transferred to the program residing cluster when computation completes.
- K files, distributed throughout the system and involved in the computation, are accessed by the modules.
- The current system state provides N available clusters.

The corresponding data structures, not shown here, would be necessary in each CSN.

2.1. The Objective Function : Total Intercluster Traffic Cost (TITC)

If $C_A = \{c_{A1},..., c_{AN}\}$ is the set of available clusters in the network, our objective is to find an execution cluster distribution[4]

$$C_D = \langle exec_1, ..., exec_M \rangle \text{ where } exec_i \in C_A,\ 1 \leq i \leq M$$

to allocate each parallel module in such a way that execution leads to a minimization of intercluster traffic according to the parallel module individual profiles, the current allocation of the program file and the current allocation of involved data files.
Our objective function deals with the following partial costs:

Initiating Cost (IC): Includes the cost to handle the user request to run the program, plus migration of parallel tasks to available clusters.
Intermodule Communication Cost (IMCC): Includes the cost to transfer messages and/or data between modules.
File Access Cost (FAC): Which includes read/write accesses from modules to data files.
Output Cost (OC): Includes the cost to transfer results from execution clusters to the initiating cluster.

$$TITC = IC + IMCC + FAC + OC$$

---

[3] A cluster is available if it is connected to the internet.

[4] The execution cluster distribution C_D, is a m-vector specifying that module i must be allocated to cluster $exec_i$.

3.1. Experiments and results

The algorithms were applied on many scenarios with diverse number of modules, clusters and files each, and a series of runs with diverse values for the corresponding input parameters were accomplished. Here we discuss the case of 10 modules, 9 clusters and 10 files.

We decided to work with an integer representation as a more natural description of the allocation problem. So, a particular position (gene) in the chromosome represented a module identifier which is allocated in the cluster identified by its value (allele). As an example when 10 modules are to be allocated in 9 available clusters, the chromosome of Fig. 2 indicates the following cluster distribution: modules 1 and 7 will be allocated in cluster 3, modules 2 and 5 in cluster 1, modules 8 and 9 in cluster 9, module 3 in cluster 4, module 4 in cluster 5, module 6 in cluster 6 and module 10 in cluster 7. Clusters 2 and 8 will not allocate any module.

cluster →	3	1	4	5	1	6	3	9	9	7
module →	1	2	3	4	5	6	7	8	9	10

Fig. 2. Chromosome structure for the cluster allocation problem

Two genetic algorithms were contrasted, a modified simple genetic algorithm (MSGA) and a combination of multiple crossovers per couple with fitness proportional couple selection (MCPC-FPCS).
The MSGA used the above indicated integer representation, uniform crossover and little-creep operators [4] for mutation.
The basic idea of FPCS [7] is to create, from the current population of individuals, an intermediate population of couples which subsequently undergoes selection for mating under MCPC those pairs showing higher fitness. To assign the fitness to the couple a criterion based on the parents fitness average ($FPCS_{AVG}$) was chosen.

The following relevant performance variables were chosen:

Best: It is the objective value of the best found individual.
Ebest = (Abs(opt_val - best value)/opt_val)100
It is the percentile error of the best found individual when compared with the known, or estimated, optimum value opt_val. It gives us a measure of how far are we from that opt_val.
Epop = (Abs(opt_val- pop mean fitness)/opt_val)100
It is the percentile error of the population mean fitness when compared with opt_val. It tell us how far the mean fitness is from that opt_val.
Gbest: Identifies the generation where the best value (retained by elitism) was found.
Time: It is the running time in seconds of the algorithm implemented on a Pentium PC (MMX of 233 MHz).

A particular benchmark case with known optimum was chosen for experimentation.

Different setting of parameters, such as crossover and mutation probabilities, population size and maximum number of generations, were used.

After running the following observations can be done:

- Both approaches find the optimum value for some setting of parameters: different crossover and mutation probabilities but equal population size of 200. This fact indicates that a greater population size improves the search process.

- MSGA finds the optimum in earlier generations.

- MCPC-FPCS finds near optimal solutions in earlier generations.

- Conventional probabilities of crossover and mutation (0.65 and 0.001) favours the MSGA approach, while lesser values for Pc and higher values for Pm (0.5 and 0.05) favours MCPC-FPCS.

- The error of the best individual (Ebest) ranges from 0.0% to 0.5% for MSGA and from 0.0% to 3.4% for MCPC-FPCS.

- The error of an average individual in the population (Epop) ranges from 0.0% to 26.6% for MSGA and from 0.6% to 27.1% for MCPC-FPCS.

- **Time**, was recorded in 2 seconds and 4 seconds for 50 and 100 generations respectively.

Summarizing MSGA seems to behave better than MCPC-FPCS for the selected suite of parameters.

But under any approach, it worth remarking that since GAs work on a population of feasible solutions, instead of providing a single optimal or near optimal solution a subset of optimal or quasi optimal solutions is always available after reaching the final generation.
This peculiar characteristic yields to fault tolerance in the model by providing a significant subset of near-optimal solutions as alternates to be used in the case the real world constraints prevent using the first selected alternate (eg. lack of availability of some cluster to be allocated). Alternate strategies can be ordered according their goodness [5].

5. Conclusions

Evolutionary computation has recently been recognized as a research field. Applications are possible in diverse areas. A broad but not exclusive categories involve scheduling, packing, routing, on line and off line control and design. Computer systems, including networks, are a promising area of applications for evolutionary computation.

The present paper introduced a model to quasi-optimally allocate available clusters of an internet, for execution of program parallel components

---

[5] *In our model we say that solution* i *shows higher goodness than solution* j *iff* $TITC_i < TITC_j$.

Two different evolutionary approaches: MSGA and MCPC-FPCS, were contrasted and despite their differences in implementation and results both, definitely provide not a single optimal solution but a set of timely optimal or near optimal solutions to the first subproblem for parallel task allocation.

Also, the flexibility of the genetic algorithm approach allows by changing parameters, such as population size and number of generations achieved, and genetic operators to tune the goodness of alternate strategies according to the accuracy and speed on response demanded by the problem. Future work is related to the incorporation of different multiple recombination approaches and incest prevention to escape from premature convergence.

Bibliography

[1] Akoka, J. – Designs of Optimal Distributed Database Systems- Proceedings of the 1st Int. Symposium on Distributed Data Bases, Paris, France, pp 229-245, 1980.

[2] Borghoff U. M. - Design of Optimal Distributed File Systems: A Framework for Research - ACM Press, Operating Systems review, Vol 26, No 4, October 1992.

[3] Chen P. P. S., Akoka J., - Optimal Design of Distributed Information Systems - IEEE Transactions on Computers c- 18:10, pp 885-889. 1969.

[4] Davis L. - Adapting Operators Probabilities in Genetic Algorithms - Proceeding of the Third International Conference on Genetic Algorithms- pag. 61-69, 1989.

[5] Eager D., Lazowska E., Zahorjan J. - Adaptive Load Sharing in Homogeneous Distributed Systems - IEEE, Transactions on Software Engineering SE-12, 662, 675, 1986.

[6] Esquivel S., Leguizamón G., Gallard R., - A Quasi-Optimal Cluster Allocation Strategy for Parallel Execution in Distributed Systems Using Genetic Algorithms, ACM Press, Operating Systems Review, USA, Vol. 29, Nr. 2, pp 82-96, April 1995.

[7] Esquivel S., Leiva A., Gallard R.: Couple Fitness Based Selection with Multiple Crossover per Couple in Genetic Algorithms. Proceedings of the International Symposium on Engineering of Intelligent Systems (EIS'98), Vol. 1, pp 235-241, La Laguna, Tenerife, Spain, February 1998.

[8] Ha□ A., Johnson T. J. - Dynamic Load Balancing through Process and Read-Site Placement in a Distributed System - AT&T Bell Tech. Journal, pp 72-85, 1988

[9] Morgan H. L., Levin K. D. – Optimal Program and Data Location in Computer Networks – Communications of the ACM, 20:5, pp 315-321, 1977.

[10] Wah B. W., - File Placement in Distributed Computer Systems- IEEE Computer 17:1, pp 23-33, Jan. 1984.