Yugoslav Journal of Operations Research
7 (1997), Number 2, 247-255

ASSIGNMENT OF DISTRIBUTED PROCESSING SOFTWARE: A COMPARATIVE STUDY

Stella SOFIANOPOULOU

Dept. of Industrial Management, University of Piraeus,
80 Karaoli & Dimitriou str., 185 34 Piraeus, Greece.

Abstract: A major issue of the operation of distributed systems is the problem of allocating a number of processes to a network of processors, with the aim of fully utilizing their potential and flexibility. This paper presents a solution to the process allocation problem from a mathematical programming point of view, employing two heuristic algorithms. The first one is an adaptation of the simulated annealing heuristic algorithm, while the second one is based on an iterative improvement procedure. The characteristics of both heuristics are briefly examined, and in the sequel both algorithms are tested on a set of random problems having characteristics similar to a real world problem.

Keywords: Distributed computing systems, process allocation, heuristic algorithms

1. INTRODUCTION

Advances in computer hardware and software have contributed to the development of distributed systems, and distributed processing is definitely an important element in today's environment. The term distributed processing has been widely used during the last decade. Enslow [3] provides a dynamic definition of the term by quoting five important properties that distributed systems should possess. These properties include multiplicity of resource components, physical distribution, existence of a high level operating system, system transparency and, finally, autonomy within the frame of cooperation.

One of the major problems encountered in the operation of distributed systems is the problem of allocating a number of processes to a network of processors, that is a set of homogeneous or heterogeneous processors with a network of communication channels connecting each processor to every other one. Every processor unit has its own local memory in order to hold data, program code, work space etc. and is constrained not to accept an excessive load of processes allocated to it. The software run on such a distributed system is also modular and is made up of a number of special

programs, known as processes, which communicate by exchanging messages. Every process when allocated to a processor occupies part of the processor's available resources (process resource requirements). It should be noted here that, certain types of processes are used more often than others and thus there is a need to include replicates of those processes that should be allocated to different procesors (replication constraints).

The communication cost associated with the message passing between processes residing on different processors is a significant criterion in the design and the operation of a distributed system. We are assuming here that communication costs between processes residing on the same processor (intra-processor communication) are negligible compared to the costs occurring when they are on different processors (inter-processor communication). Thus, it can be clearly seen that different allocations of processes to processors result in a different amount of inter-processor overhead. Our target, therefore, is to determine an allocation that has the minimum inter-processor (or maximum intra-processor) message passing.

The purpose of this paper is to examine and compare two heuristic algorithms for the solution to the process allocation problem. A brief presentation of the model which describing the problem mathematically is presented. The simulated annealing heuristic as well as the iterative transformation procedure are briefly described. The two algorithms are compared and their performance is evaluated using two different comparison methods. Conclusions are drawn in the last section.

2. MATHEMATICAL FORMULATION

The Process Allocation Problem (PAP) presented in [9] deals with the allocation of a number N of communicating processes to a network of processors. Each of the identical processors has its own dedicated memory and a complete network of communicating channels that links it to all other processors. Different types of memory constraints as well as capacity constraints on each processor are also present in the model. Associated with the exchange of messages between processes i and j is communication cost c_{ij}, while the optimization criterion in the model is the minimization of the message passing cost between processes residing in different processors. The problem is formulated as following:

$$mimize\ z = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} c_{ij}(1 - x_{ij}) \tag{1}$$

so that

$$\sum_{i=1}^{k-1} r_j x_{ik} + \sum_{j=k+1}^{N} r_j x_{kj} \le R - r_k \quad k = 1,\ldots,N \tag{2}$$

$$\begin{aligned} & x_{ij} + x_{ik} - x_{jk} \le 1 \\ & x_{ij} - x_{ik} + x_{jk} \le 1 \\ & -x_{ij} + x_{ik} + x_{jk} \le 1 \end{aligned} \qquad \begin{aligned} & i = 1,\ldots,N-2, \\ & j = i+1,\ldots,N-1, \;\; k = j+1,\ldots,N \end{aligned} \tag{3}$$

$$x_{ij} = \begin{cases} 1, \text{ if processes } i \text{ and } j \text{ are assigned to the same processor} \\ 0, \text{ otherwise,} \qquad\qquad i = 1,\ldots,N\text{-}1, \; j = i+1,\ldots,N \end{cases} \tag{4}$$

Constraints (2) refer to the resource limitations of the model. Assuming that R is the resource availability (e.g. memory, capacity etc.) of each processor, then these constraints ensure that the sum of the resource requirements r_i of all processes i connected to process k, i.e. assigned to the same processor as process k, does not exceed quantity R-r_k, which is the spare capacity of the processor after having allocated process k to it. Constraints (3) are imposed to preserve the triangular structure of the model. They ensure that if x_{ij}=1 and x_{ik}=1, then processes j and k should be also connected, i.e. x_{jk}=1.

The exact solution to the integer programming problem (1) through (4) provides the optimum allocation of processes to processors. However, obtaining the exact optimum in an integer programming problem in practice is often either impossible or simply unattractive. Since the optimal solution to the PAP becomes quite expensive in terms of computing time, researchers have used different heuristic approaches that produce suboptimal solutions. A survey of heuristic assignment algorithms for the PAP and their performance analysis is presented by Price and Krishnaprasad [8], while an efficient algorithm based on simulated annealing and adapted to PAP has been developed by the author [10], and is briefly presented in the next section.

3. SIMULATED ANNEALING METHOD

The simulated annealing algorithm is a heuristic method based on randomization techniques. The method has been successfully applied to combinatorial problems in computer systems design [5], the solution of the travelling salesman problem [4], the solution of the quadratic assignment problem [2], the minimization of message passing cost in the PAP [10] and the machine cell formation [11] in a manufacturing context.

The basic idea is to generate random displacements from any current feasible solution, and accept as the new current solution not only solutions which improve the objective function, but also some which do not improve it; the latter ones are accepted with probability $\exp(-\Delta f / T)$, depending on the amount of deterioration Δf of the objective function and a tunable parameter T (the temperature).

The two major components required in the implementation of simulated annealing are:

1. A perturbation mechanism for generating random neighbor solutions.
2. An annealing schedule consisting of

a) A finite decreasing sequence of values T_t, t=1,2,... for the control parameter T of temperature; i.e. initial and final values and a rate of decrease.

b) The number of solutions L_t attempted at each temperature T_t as temperature decreases.

The algorithm iterates until the stopping temperature value is reached or no feasible solution has been found during L_t attempts.

In the present implementation, the neighborhood generation scheme consists of randomly selecting and connecting two processes p and q which are not interconnected, i.e. the corresponding variable x_{pq} which is 0, is made equal to 1. This move has some implications which involve disconnecting process p from all processes to which it was previously connected and connecting it to all processes to which q is already connected (triangular constraints (3) of section 2). Obviously, this move is carried out only if the capacity constraints (2) are not violated.

The initial (and final) value of T is determined using a small number of pilot runs before the actual annealing process begins, at an appropriately high (low) value so that almost all candidate solutions that deteriorate the objective function are accepted as the current solution with a probability of 0.95 or more (0.05 or less). The cooling schedule, i.e. the rate of decrease of temperature T is very crucial for the successful application of the algorithm. If the rate of temperature decrease is too high then the algorithm leads to local optimum solutions, while if it is too low, CPU time is wasted. In this work the cooling schedule suggested in [1] is adopted where T_t is updated by:

$$T_{t+1} = T_t / \{1 + [\, T_t \ln(1+\delta\,) / \; 3\sigma_t \,]\} \qquad (5)$$

where δ =0.1 and σ_t is the standard deviation of the objective function values of the solutions examined at T_t.

The number of solutions L_t attempted at each temperature T_t is set to $N(N-1)/2$. This parameter setting is adopted taking into account that the number of all possible moves that can lead a current solution to a neighboring one (i.e. making variable x_{pq}, which is currently set to 0, equal to 1) is at most $N(N-1)/2$ (actually there are much fewer since capacity and triangular constraints should be satisfied).

Having defined the above ingredients of the simulated annealing and an initial feasible solution to the PAP, one can apply the algorithm. Obviously, the number of iterations to be performed depends on the size of the problem. For the sake of completeness it should be mentioned that a starting feasible solution to the PAP is formed by randomly choosing processes and grouping them together until some capacity constraint is violated. If this is the case, the next randomly chosen process starts forming a second group and this procedure is continued until all processes are grouped.

4. ADAPTED ITERATIVE TRANSFORMATION PROCEDURE

Another heuristic approach which has been used to tackle the PAP is an "iterative transformation procedure" proposed by Price [6] and Price and Garner [7]. This improvement procedure is based not only on the communication costs between processes residing on different processors, but also on the execution cost of each process on each processor. Since this second kind of cost is not present in our version of the PAP, the iterative transformation procedure has been properly modified. The method begins with a feasible assignment of processes to processors. Then at each iteration all possible reassignments of processes are evaluated taking into account the communication costs, and every time the most advantageous one is chosen. The procedure continues until no better reassignment can be found. The communication cost of an assignment $Y=\| y_{ij} \|_{NxM}$ is calculated as:

$$\sum_{i=1}^{N-1} \sum_{k=i+1}^{N} c_{ik} - \sum_{i=1}^{N-1} \sum_{j=1}^{M} \sum_{k=i+1}^{N} c_{ik} y_{ij} y_{kj} \tag{6}$$

where N is the number of processes and M the number of processors. Actually, M represents the number of processors used in the initial feasible assignment of processes, i.e. in the assignment from which the iterative transformation procedure starts. Obviously, M also represents the maximum possible number of processors to be utilized. Perhaps, it should be emphasized here that in the formulation (1)-(4) proposed in section 2 the number of processors to be utilized is not known *a priori* (refer also to [9]).

The reassignment of processes to proccessors is attained by a transformation in the assignment matrix Y, where:

$$y_{ij} = \begin{cases} 0 \\ 1 \end{cases} \qquad \text{for all } i,j \tag{7}$$

and

$$\sum_{j=1}^{M} y_{ij} = 1 \qquad \text{for all } i \tag{8}$$

The element $y_{ij} = 1$ of the matrix Y denotes that process i is assigned to processor j. The transformation $T : A \rightarrow A$ of the set of all assignments A is performed as follows:

1. Given an assignment matrix defining an allocation of processes to processors, every process is temporarily allocated to each one of the processors and a penalty is computed. The penalty is equal to the communication cost of the particular process with all other processes when they are assigned as indicated in the assignment matrix Y.

2. The minimum penalty is determined for each process. If the minimum penalty corresponds to the assignment of the particular process as it is in the assignment matrix Y, then the iterative procedure stops. If, however, there is a process which when allocated to a different processor (from the one that is currently assigned) produces a smaller communication overhead in the total cost, a new solution-assignment is determined. If there is more than one such process, then the one that produces the smallest overhead in the total cost is chosen.

3. The assignment matrix Y is modified by relocating the process determined in Step 2 and transformation T is repeated.

Finally, it should be noted that the iterative transformation procedure can be modified and extended to guarantee convergence to a global optimum, but at a considerable computational cost [6]. This modification consists (after completion of the above described procedure) of the evaluation of all two-step transformations, that is, all simultaneous reassignments of two processes. The procedure then continues with all three step reassignments and so on until we reach the transformation that simulataneously reassigns all N processes. Of course, this generalization of the procedure corresponds to a complete enumeration of all possible solutions.

5. COMPARATIVE STUDY

The computational results presented concern fourteen test problems. These data sets were randomly produced so as to resemble an original sample set of data provided by a telecommunications laboratory. This sample data set refers to an instance of the PAP with 12 processes, each having a particular occupancy, code- and data-storage requirement. Results obtained with the annealing method are compared with those of the "iterative transformation procedure". The test problems used include no replicates. Because of the absence of execution costs the initial feasible assignment was obtained using exactly the same procedure we have employed in the annealing method.

Two approaches where adopted in this comparative study. In the first, the iterative heuristic of Price was run 520 times, which is approximately equivalent, as far as CPU time is concerned, to ten runs of 1500 iterations of the annealing heuristic. The corresponding best solutions and average gaps between the optimum (computed employing an implicit enumeration method developed by the author [9]) and the heuristic values produced are compared. As can be seen from Table 1, where the best solutions found, the gap values, the CPU times, on a VAX 11/780 system, and the optimum objective function values are presented, the annealing heuristic has a better performance than the iterative transformation one. Out of the fourteen test cases tried, the simulated annealing algorithm achieved the global optimum in thirteen, while the iterative heuristic achieved it in only five. Also in almost every case with the annealing heuristic the average gap did not exceed 2% (with the exception of only two cases out of fourteen) while the corresponding average gap produced by the iterative transformation procedure where more than 5%.

In the second approach the expected number of runs (and hence CPU time) required in order to obtain at least one successful result with probability 95%, is evaluated for both heuristics and compared. Note that any solution within 1% of the optimum is considered to be a successful result. Assuming that for each test case the probability of success of a single run is p (empirically estimated from a large number of runs), the probability of performing N runs without a single success is $(1\text{-}p)^N$. Setting this probability equal to 0.05 (i.e. 1-0.95) the required number of runs N with at least one success is given by:

$$N = \log(1\text{-}0.95)/\log(1\text{-}p) \tag{9}$$

Then, the corresponding total CPU time required for one success is computed by multiplying N by the average per run CPU time. It should be mentioned that the probability p of achieving a successful result and the average per run CPU time is evaluated for the annealing heuristic and the iterative transformation procedure based on 100 and 520 runs respectively.

Table 1

Data set	Annealing Heuristic (*)			Iterative Transformation Procedure (**)			Optimum
	Best Solution	Gap (%)	CPU (secs)	Best Solution	Gap (%)	CPU (secs)	
1	436	1.491	102.37	436	6.383	103.54	436
2	484	0.847	106.24	485	5.760	125.28	484
3	489	0.470	88.96	491	7.157	125.01	489
4	496	0.766	88.49	500	5.360	126.14	496
5	501	0.739	98.06	505	6.301	120.09	501
6	496	1.310	107.45	498	3.983	103.46	496
7	454	1.145	103.51	454	5.329	228.39	454
8	467	2.227	87.36	467	8.098	238.30	467
9	475	2.336	89.24	475	10.49	100.56	475
10	520	1.340	108.18	515	4.072	316.39	515
11	467	0.257	107.75	469	5.317	123.54	467
12	511	0.196	112.93	516	5.137	123.68	511
13	465	1.548	103.78	470	7.262	114.37	465
14	460	1.891	93.41	465	8.347	118.47	460

(*) 10 runs of 1500 iterations
(**) 520 runs

The calculated required number of runs and total CPU time for the annealing heuristic for 1000, 1500 and 2000 iterations respectively and for the iterative transformation procedure are demonstrated in Table 2. In all cases (but three) the annealing heuristic required a smaller amount of CPU time in order to guarantee (with probability 95%) at least one successful run out of N. It is interesting to note that one expects to require a relatively lower number of runs in order to achieve one successful

result, as the number of iterations of the annealing heuristic increases. The three major columns of Table 2 for the 1000, 1500 and 2000 iterations of the annealing heuristic do exhibit in general such a behavior. There are, however, a few exceptions (e.g. data sets 8, 13 and 14) that are due to the probabilistic nature of the annealing heuristic. The empty spaces in the last major column of Table 2 indicate that none of the 520 runs of the iterative transformation procedure produced a successful result.

Table 2

Data set	Annealing Heuristic (*)						Iterative Transformation Procedure (**)	
	1000 iterations		2000 iterations		3000 iterations			
	No of runs (+)	CPU (secs)	No of runs (+)	CPU (secs)	No of runs (+)	CPU (secs)	No of runs (+)	CPU (secs)
1	20	131.40	14	140.70	11	144.10	778	171.16
2	6	41.82	4	41.44	3	41.91	388	93.12
3	5	29.65	5	44.40	3	36.48	172	41.28
4	4	23.96	2	18.66	2	24.70	1556	373.44
5	4	25.72	4	39.80	2	26.74	388	93.12
6	7	48.23	4	41.36	4	53.88	41	8.20
7	12	84.72	9	92.70	7	97.86	518	233.10
8	15	87.30	22	199.10	9	104.04	110	51.70
9	41	223.45	20	164.00	10	101.90	-	-
10	6	42.06	4	44.96	3	42.57	388	240.56
11	2	13.98	2	22.06	1	14.31	1556	373.44
12	1	7.16	1	10.94	1	14. 59	258	61.92
13	10	65.30	11	101.31	7	84.77	-	-
14	15	95.55	14	129.36	16	197.92	1556	357.88

(*) results based on 100 runs
(**) results based on 520 runs
(+) number of runs required to obtain at least one "success" with 95% probability

6. CONCLUSIONS

The assignment problem of distributed processing software has been briefly presented. The simulated annealing algorithm and an iterative transformation procedure adapted to the PAP have been examined and compared. Two different approaches where used to compare the efficiency of the two heuristic algorithms. The computational results obtained show the superiority of the simulated annealing method.

REFERENCES

[1] Aarts, E., and Korst, J., *Simulated Annealing and Boltzmann Machines*, Wiley, Chichester, 1990.

[2] Burkard, R.E., and Rendl, F., "A thermodynamically motivated simulation procedure for combinatorial optimization problems", *European Journal of Operational Research*, 17 (1984) 169-174.

[3] Enslow, P.H., "What is a 'distributed' data processing system?", *Computer*, 11 (1978) 13-21.

[4] Golden, B.L., and Skiscim, C.C., "Using simulated annealing to solve routing and location problems", *Naval Research Logistics Quarterly*, 33 (1986) 261- 279.

[5] Kirkpatrick, F., Gelatt C.D. jr., and Vecchi, M.P., "Optimization by simulated annealing", *Science*, 220 (1983) 671-680.

[6] Price, C.C., "The assignment of computational tasks among processors in a distributed system", *Proceedings AFIPS National Computer Conference*, 1981, 291- 296.

[7] Price, C.C., and Garner, J. M., "Task assignment modelled as a quadratic programming problem". Report DMS-7, Dept. of Maths. and Stats., Stephen F. Austin State University, Texas, 1983.

[8] Price, C.C., and Krishnaprasad, S., "Software allocation models for distributed computing systems", *Proceedings 4th International Conference on Distributed Computing Systems*, 1984, 40-48.

[9] Sofianopoulou, S., "Optimum allocation of processes in a distributed environment: a process-to-process aproach", *Journal of Operational Research Society*, 41 (1990) 329-337.

[10] Sofianopoulou, S., "Simulated annealing applied to the process allocation problem", *European Journal of Operational Research* 60 (1992) 327-334.

[11] Sofianopoulou, S., "Application of simulated annealing to a linear model for the formulation of machine cells in group technology", *International Journal of Production Research*, 35 (1997) 501-511.