Proceedings of 2009 12th International Conference on Computer and Information Technology (ICCIT 2009)
21-23 December, 2009, Dhaka, Bangladesh

Heuristic Algorithm of the Multiple-Choice Multidimensional Knapsack Problem (MMKP) for Cluster Computing

Md. Iftakharul Islam, Md. Mostofa Akbar

Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology (BUET), Dhaka-1000, Bangladesh
tamim@csebuet.org, mostofa@cse.buet.ac.bd

Abstract

This paper presents two heuristic algorithms of the MMKP (a variant of 0-1 knapsack problem) for cluster computing. We present an architecture of a cluster, such that algorithm requires small message passing. The algorithms divide the problem among computational nodes. Each node solves its sub problem using a sequential heuristic. This naïve divide and conquer approach cannot achieve good revenue. The revenue is the value achieved by the solution of MMKP. To improve the revenue, it accumulates the unused resources from every node, and assigns to the node, which gives maximum revenue over all nodes. This is the residue exploitation (RE) strategy. The solution quality can be improved by a novel resource-division policy rather than equal division. The policy divides the resource among all nodes such that total revenue increases. A sequential heuristic calculates the solution incrementally for different amounts of resource capacity, and the best combination is taken as the solution. This is the resource adjustment (RA) strategy. We experiment the algorithm using MPI (Message Passing Interface). The proposed algorithms show encouraging results.

Keywords: Admission control architecture, Heuristic algorithm, Knapsack problem, Parallel processing.

I. INTRODUCTION

This paper will present two distributed heuristics of the MMKP for cluster computing. In the MMKP, a set of groups is given. Each group has some items. Each item, requiring multiple resources, gives a nonnegative revenue. The problem is to select an item from every group, such that the resource constraints of knapsack are not violated, and the revenue is maximized. Let there be n groups. Each group is denoted as $J_l, l = 1, \dots, n$. Group J_l has r_l items. Each item $j, j = 1, \dots, r_l$ of group J_l has a value v_{lj}, and requires m resources $W_{lj} = (w_{lj}^1, w_{lj}^2, \dots, w_{lj}^m)$. $C = (C^1, C^2, \dots, C^m)$ be the resource bound of the knapsack. The objective of the MMKP is to pick exactly one item from each group for maximum total revenue of the collected items, subject to the resource constraints of the knapsack. That is, the problem is to find $max \sum_{l=1}^{n} \sum_{j=1}^{r_l} x_{lj} v_{lj}$ such that $\sum_{l=1}^{n} \sum_{j=1}^{r_l} x_{lj} w_{lj}^k \leq C_k$ for $k = 1,2, \dots, m$ where $x_{lj} \epsilon \{0,1\}$ and $\sum_{j=1}^{r_l} x_{lj} = 1$.

Many real-world problems, for example, admission control and resource scheduling for multimedia system can be formulated in terms of MKPP [8]. We will propose a cluster architecture, such that the algorithm requires small message passing. We will show the architecture in the context of admission control of multimedia server. The remainder of the paper is organized as follows. In Section 2, we present different solutions for MMKP. Section 3 describes the cluster architecture. Section 4 presents the algorithms. In Section 5, we evaluate the performance of the algorithms. Section 6 concludes the paper.

II. RELATED WORKS

To solve the MMKP, an approach was designed based on Lagrange multipliers [10]. An algorithm has been proposed (named HEU) based on the aggregate resources [9]. A preprocessing and post processing is augmented to the HEU is called M-HEU is presented in [1]. It helps the algorithm to go out of local optima. An incremental version of M-HEU, called I-HEU, is also proposed in [1]. Guided Local Search (GLS) algorithm is a recent approach for solving MMKP [5]. The Linear Programming Relaxation (LPR) of MMKP, called HMMKP, is described in [4]. Another heuristic, based on convex hull, is proposed in [2]. Later, Reactive Local Search (RLS) based approach is used for MMKP [6]. Dividing the computation of M-HEU among multiple processors, a parallel heuristic for MMKP is proposed for shared memory architecture [11]. This paper presents heuristic solutions of MMKP for distributed memory architecture.

III. CLUSTER ARCHITECTURE

In the section, we discuss a cluster computing architecture for solving the MMKP, in an admission controller of a multimedia server. In a trivial cluster architecture, all the requests come to a server. The server takes the help of other nodes to solve the admission control problem. If the server divides the problem among all nodes, huge message passing will be required. In our proposed architecture, we will divide the problem while posing to the server. So, no overhead will be required for dividing the problem. Fig. 1 illustrates the architecture.

978-1-4244-6284-1/09/$26.00 ©2009 IEEE

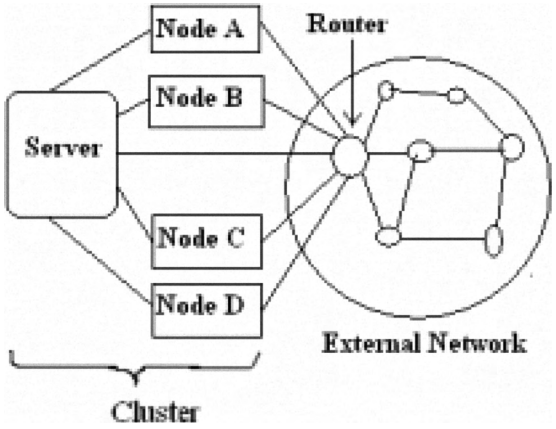

Fig. 1. Cluster computing architecture of an admission controller for multimedia server.

Node A, B, C, D, and the server form the cluster. A router is connected to every node using point to point links. The router intelligently divides the MMKP among all nodes. When a request (also called group in terms of MMKP) arrives at the router, router destines it to a node in a round robin fashion. So, every node has almost equal number of groups. A parallel-heuristic algorithm solves the whole problem. At the end of the computation, every node sends the list of selected items to the server. Thus, the messages include only one item per group instead of a whole group. In this way, we reduce the cost of message passing significantly.

Besides the network architecture, many other issues affect the performance. Some issues of ethernet cluster are studied in [7]. In our study, we adopt static task assignment policy. Scattered subdivision is chosen for dividing the problem among nodes. In scattered subdivision, packet i is assigned to the node $i \bmod p$, where p is the total number of nodes.

IV. ALGORITHM

We propose two algorithms. In the first algorithm, only Residue Exploitation (RE) is used. In the second algorithm, both Resource Adjustment (RA) and Residue Exploitation (RE) are used. Both of the algorithms divide the problem among nodes. The algorithms are abbreviated as DCRE and DCRARE respectively, where 'DC' stands for Divide and Conquer.

A. DCRE ALGORITHM

The steps of the DCRE algorithm are the following:

1. Server divides resources equally (almost) among all nodes.
2. Each node calculates a local solution using a sequential heuristic.
3. In most of the cases, some resources are not totally used while one or more other resources are finished. Each node calculates the unused resources.
4. Each node sends the amount of unused resources to the server.
5. The server calculates the total unused resources, and sends it to every node.
6. Each node tries to increase its revenue using the unused resources.
7. Each node calculates its revenue increment based on the unused resources, and sends it to the server.
8. Server finds the node, which obtains maximum revenue increment.
9. The solution of that node is updated.
10. Finally, each node sends its solution to the server.

B. DCRARE ALGORITHM

The algorithm divides the resources among nodes. Each node calculates local solutions for the different amounts of resources incrementally. Then, an amount of resources is chosen for each node that gives maximum total revenue. Finally, it collects the unused resources, and allocates to a node, as it is done in DCRE.

To demonstrate the allocation of resources, we introduce two parameters: λ (a positive odd number) and δ ($0< \delta<1$). λ indicates the number of allocation levels of the assigned resources. δ indicates the difference between the two consecutive allocation levels. If we consider $\lambda=7$ and $\delta=0.05$, then the fractions of allocation are: 0.85, 0.90, 0.95, 1.0, 1.05 and 1.10, 1.15. The fraction to be allocated at level t is indicated by the following expression:

$$1.0 + \left(t - \left\lceil \frac{\lambda}{2} \right\rceil\right) * \delta \quad \text{where } (1 \le t \le \lambda)$$

λ is considered as an odd value with the following levels: i) Level $1, \ldots, \left\lceil \frac{\lambda}{2} \right\rceil - 1$ indicates the pessimistic allocation. Each of these levels is less than 1.0 in terms of fraction of assigned resources. ii) Level $\left\lceil \frac{\lambda}{2} \right\rceil$ indicates the exact allocation. It is analogous to the fraction 1.0. iii) Level $\left\lceil \frac{\lambda}{2} \right\rceil + 1, \ldots, \lambda$ indicates the optimistic allocation. Each of the levels is higher than fraction 1.0.

The algorithm starts with the minimum fraction, and increases the fraction by δ amount in every iteration. In the first iteration, local solution is obtained by a sequential heuristic. In the next iterations, the solution of the previous iteration is upgraded, taking it as the initial solution. The algorithm is presented in Algorithm 1.

The procedure *allocate_resource* is described in procedure 1. Line 1of *allocate_resource* initializes the solution for fraction 1.0. Line 4 and 6 finds the node, which are most suitable for up-gradation and down-gradation respectively. Line 10-12 update the solution to increase revenue.

Time complexity of the algorithms depends on the sequential algorithm, which is used to find local solutions. Both DCRE and DCRARE uses divide and conquer strategy. So, both are very high throughput algorithm. DCRARE calculates local solutions incrementally for λ times. It calculates an initial

solution, and then upgrades it. Time complexity will not increase considerably for this incremental fashion. Time complexity of *allocate_resource* is $O(\lambda p^2)$.

Algorithm 1: DCRARE (λ, δ)

1. Server divides the resources among nodes
2. Each node calculates a minimum fraction of assigned resource which is discussed before.
3. Each node calculates the solutions for each fraction of resources incrementally.
   - 3.1. Calculate the solution for the initial fraction of resource using a sequential heuristic. The solution of each iteration will be the input of the next iteration.
   - 3.2. Calculate the solutions for subsequent fractions of resources. It takes the solution of previous step as an initial solution, and upgrades it. This step will be repeated for $\lambda - 1$times.
   - 3.3. Each node forms a $1 \times \lambda$ matrix where each entry holds the revenue for the different fractions.
   - 3.4. This matrix is send to the server.
4. Server forms a $p \times \lambda$ matrix M from the $1 \times \lambda$ matrix of each node.
5. Call the procedure *allocate_resource* (M)
6. Solution obtained by *allocate_resource*(M) is updated.
7. Apply residue exploitation procedure of DCRE.

Procedure 1: allocate_resource (M)

Input: M is a $p \times \lambda$ matrix , M_{it} indicates the revenue, which is earned by the ith node for level t.

Output: $H = (H_1, \ldots, H_p)$ where M_{iH_i} holds revenue of the ith node that gives better global revenue.

1. $H_i \leftarrow \lceil \lambda/2 \rceil$ ***for*** $1 \le i \le p$
2. **for** $t = 1, \ldots, \lceil \lambda/2 \rceil - 1$ ***do***
3. ***while(true) do***
4. $a_0 \leftarrow \underset{1 \le a \le p}{argmin} \{M_{aH_a} - M_{a(H_a - t)}\}$
5. $a_0\_revenue \leftarrow M_{a_0 H_{a_0}} - M_{a_0 (H_{a_0} - t)}$
6. $b_0 \leftarrow \underset{1 \le b \le p, b \ne a_0}{argmax} \{M_{b(H_b + t)} - M_{bH_b}\}$
7. $b_0\_revenue \leftarrow M_{b_0 (H_{b_0} + t)} - M_{b_0 H_{b_0}}$
8. ***if***$(a_0 = null \text{ or } b_0 = null)$ ***then***
9. ***break;***
10. ***if*** $(b_0\_revenue > a_0\_revenue)$***then***
11. $H_{a_0} \leftarrow H_{a_0} - t$
12. $H_{b_0} \leftarrow H_{b_0} + t$
13. ***else***
14. ***break;***
15. ***end while.***
16. ***end for.***
17. ***return*** $(H_1, \ldots, H_p)$

V. EXPERIMENTAL RESULTS

PVM (Parallel Virtual Machine) and MPI (Message Passing Interface) facilitate the implementation of parallel applications on multiprocessor environments (especially for distributed memory architecture). PVM supports heterogeneity, portability, and fault tolerance while sacrificing performance. MPI does not provide much flexibility as PVM, but demonstrates high performance [3]. To achieve better performance, we have implemented our algorithm using MPI. We tested our algorithms on a cluster of thirteen nodes connected by a Myrinet GM network. Every node is a Pentium IV, 1.7 GHz processor, 128 Mb RAM with Linux operating system. The algorithms have been coded using C programming language. We vary the number of nodes to observe the effect. We have tested the two algorithms on the benchmark instances, found in http://www.laria.u-picardie.fr/hifi/OR-Benchmark/MMKP/MMKP.html. Here, I01,… I06 are small problems, and I07, I08, … , I13, Ins01, Ins02, … , Ins20 are medium and large problems. Our algorithms need a sequential heuristic to solve MMKP locally. We have taken MRLS as the sequential heuristic, as it demonstrates better performance among all the existing heuristics and standard package, for example, Cplex Solver [6]. Table I shows the parameters related to the performance of MRLS with their associated values for better revenue [6].

Table I: Parameters of sequential heuristic

Number of iterations	Number of solutions can be upgraded	Length of memory list for detection of cycling
50n	5	$[2n, 2n + 10]$

Limited experiment shows that as number of node increases, computation time decreases with slight minor solution. For small problem (I01,… I06), search space reduces quickly with increase of the number of nodes (p). So, revenue also reduces quickly. For larger problems, search space does not become too small after dividing into sub problems. So, solution quality does not reduce much. It supports the reason of parallelization that we need parallelization mainly for large problem. For small problem, we should use sequential algorithm rather than DCRE or DCRARE.

Fig. 2 shows how the number of nodes affects solution quality and time requirement. The result of DCRE algorithm is represented as a dotted line. The data with one node represents the result of sequential heuristic MRLS. The data with more than one node represents the result of parallel heuristics DCRE and DCRARE. This experiment is conducted on large and medium sized problems (I07, I08, … , I13, Ins01, Ins02, … , Ins20) of the benchmark. Here, the revenues and times are taken as average. For DCRE, as p increases, computation time reduces with reduced quality of solution. From the figure, it is clear that it is highly scalable algorithm. As p increases, the number of messages increases. These messages create heavy traffic in the server. As a result,

communicational overhead increases with the increase of p. Again, residue exploitation step is common in all case (do not depend on number of node). So, time requirement will not decrease linearly with the increase of p. Note that, the revenue is very close to the sequential heuristic, up to $p = 5$, and the computation time is much less than that. But, after a certain level, revenue decreases exponentially. The level depends on the size of problem. So, DCRE gives a good quality solution only up to certain number of node. The choice of p depends on the required revenue and computation time.

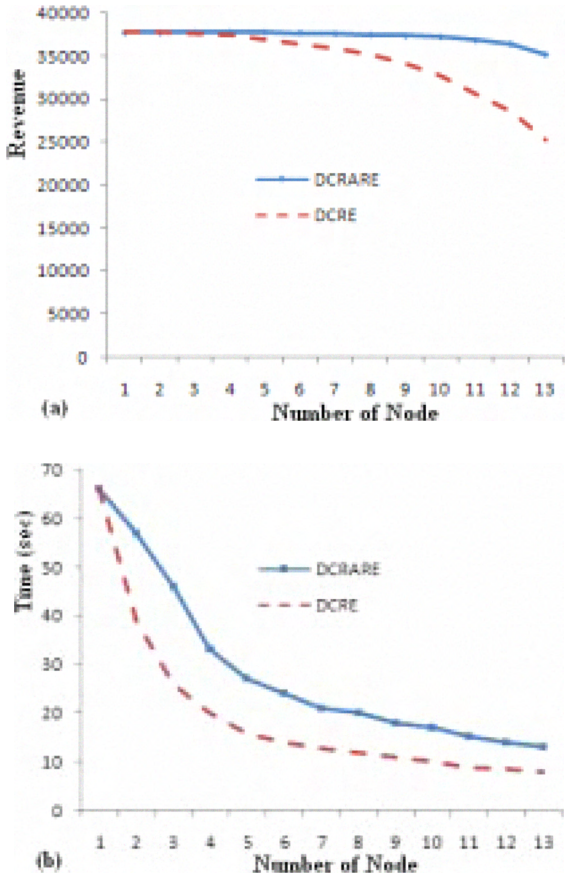

Fig. 2. (a) Average revenue Vs number of nodes (b) Average time Vs number of nodes

In DCRARE approach, we calculate solutions for different amount of resource capacity cumulatively. The solution of each iteration is treated as the initial solution for next iteration. For this cumulative process, time complexity will not increase significantly. Let the initial fraction be η. Note that, η is calculated as a function of λ and δ. Limited computational result shows that if η is less than 0.9, it gives an inferior solution with small resource capacity. It also increases computation time. If η is very close to 1.0, then it is almost equivalent to DCRE. So, value of η should be taken 0.90. This imposes a constraint on the value of λ and δ. Table II shows the revenues and time requirements for different values of λ and δ. This experiment is done for different values of p. The table shows a tradeoff between running time and revenue. For small value of λ, the algorithm solves for small number of combinations. This will increase throughput but decrease revenue. For (λ, δ) = (21, 0.01), revenue increase with increasing running time significantly. A reasonable choice is (λ, δ) = (11, 0.02).

As p increases, the combination of local solutions increases. This helps to find a better global solution. Again, search space decreases with increase of p. So, revenue of local solution decreases with increase of p. Thus, two conflicting effect falls on a global solution with increase of p. Fig. 2 describes how the revenue and computation time changes with the number of node. The result of DCRARE algorithm is represented as a solid line. As p increases, revenue increases (very small amount) until $p = 4$. In this situation, size of the sub problem does not become too small. Rather, a better global solution can be found from the combination of local solutions. So, revenue increases. Thus, DCRARE gives better revenue than MRLS. If we increase p more, revenue decreases slowly as the size of the sub problems reduces. From the figure, it is clear that revenue achieved by DCRARE is much better than DCRE. Again, computation time also decreases with increase of p. It is a highly scalable algorithm, but it's time requirement is greater than DCRE.

Table II. The behavior of algorithm DCRARE for different value of λ, δ

λ	δ	Avg. Time (sec)	Avg. Revenue
21	0.01	55	60011
11	0.02	42	59998
7	0.03	36	59636
5	0.04	32	59195

VI. CONCLUSION

Both DCRARE and DCRE achieve good quality of solution within very small time. DCRARE achieves more revenue than DCRE, but requires more time than it (DCRE). DCRARE and DCRE can be employed to develop an admission control and resource scheduling framework for distributed multimedia server. The algorithms are the first algorithm of the MMKP for cluster computing. Many issues of these distributed algorithms are still open. In DCRE, the solution quality reduces greatly for large number of nodes. Research should be made to address this problem. We experiment the algorithms with a cluster of thirteen nodes. A survey should be conducted with a large cluster for extensive set of problem instances. These algorithms should be experimented for several real life applications. We experiment our algorithms with static task assignment strategy. Research should be conducted on dynamic task

assignment and dynamically varying the number of nodes.

VII. ACKNOWLEDGEMENT

The authors would like to thank the Computer Science and Engineering (CSE) department of BUET (Bangladesh University of Engineering and Technology) for the support to conduct the experiment.

REFERENCES

[1] M. M. Akbar, E. G. Manning, G. C. Shoja and S. Khan, "Heuristic Solutions for the Multiple-Choice Multi-Dimension Knapsack Problem", International Conference on Computational Science, (ICCS 2001) vol. 2074 of Lecture Notes in Computer Science, pp 659-668, Springer, 2001.

[2] M.M. Akbar, M. S. Rahman, M. Kaykobad, E.G. Manning, and G.C. Shoja, "Solving the Multidimensional Multiple-Choice Knapsack Problem by Constructing Convex Hulls", Computers & Operations Research, Elsevier Science, 2006.

[3] G. A. Geist, J. A. Kohl, P. M. Papadopoulos, "PVM and MPI: a Comparison of Features," Calculateurs Paralleles Vol. 8 No. 2 (1996).

[4] R. Parra-Hernandez and N. Dimopoulos, "A new Heuristic for Solving the Multi-choice Multidimensional Knapsack Problem", IEEE Transaction on Systems, Man and Cybernetics, Part A: Systems and Humans,2005.

[5] M. Hifi, M. Michrafy and A. Sbihi, "Algorithms for the multiple-choice multi-dimensional knapsack problem", Journal of the Operational Research Society, vol. 55, pp. 1323-1332, 2004.

[6] M. Hifi, M. Michrafy and A. Sbihi, "A Reactive Local Search-Based Algorithm for the Multiple-Choice Multi-Dimensional Knapsack Problem", Computational Optimization and Applications, 33, 271–285, 2006.

[7] M. Hamdi, Y. Pan, B. Hamidzadeh, F. M. Lim, "Parallel Computing on an Ethernet Cluster of Workstations: Opportunities and Constraints", The Journal of Supercomputing, 13, 111-132,1999.

[8] S. Khan, "Quality adaptation in a multisession multimedia system: Model, algorithms and architecture," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, Canada, 1998.

[9] S. Khan, K. F. Li, E. G. Manning, M. M. Akbar. "Solving the Knapsack Problem for Adaptive Multimedia System", Studia Informatica, 2002.

[10] M. Moser, D. P. Jokanovic and N. Shiratori, "An Algorithm for the Multidimensional Multiple-Choice Knapsack Problem", IEICE Transactions on Fundamentals of Electronics, pp 582-589, Volume 80(3), 1997.

[11] A. Z. M. Shahriar, M. M. Akbar, M. S. Rahman, and M. A. H. Newton, "A Multiprocessor based Heuristic for Multi-dimensional Multiple-Choice Knapsack Problem". The Journal of Supercomputing, Volume 43, 2008.