ISSN: 2229-6956(ONLINE)
DOI: 10.21917/ijsc.2012.0045

PARALLEL IMPLEMENTATION OF CROSS-LAYER OPTIMIZATION - A PERFORMANCE EVALUATION BASED ON SWARM INTELLIGENCE

Vanaja Gokul[1] and Susan Elias[2]
Department of Computer Science & Engineering, Sri Venkateswara College of Engineering, Tamil Nadu, India
E-mail: [1]vanajagokul@gmail.com and [2]susanelias@gmail.com

Abstract
In distributed systems real time optimizations need to be performed dynamically for better utilization of the network resources. Real time optimizations can be performed effectively by using Cross Layer Optimization (CLO) within the network operating system. This paper presents the performance evaluation of Cross Layer Optimization (CLO) in comparison with the traditional approach of Single-Layer Optimization (SLO). In the parallel implementation of the approaches the experimental study carried out indicates that the CLO results in a significant improvement in network utilization when compared to SLO. A variant of the Particle Swarm Optimization technique that utilizes Digital Pheromones (PSODP) for better performance has been used here. A significantly higher speed up in performance was observed from the parallel implementation of CLO that used PSODP on a cluster of nodes.

Keywords:
Cross Layer Optimization, Particle Swarm Optimization, Digital Pheromones, Network Utilization, Parallel Implementation

1. INTRODUCTION

Cross Layer Optimization (*CLO*) can contribute to an improvement in the network performance under various operational conditions [1, 3] and is emerging to be an interesting and challenging research area particularly in wireless network for multimedia communication [16]. Efficient utilization of resources is the primary goal of *CLO* and it is effectively being used currently in wired communication systems. Wireless communication systems that are dynamic in nature require optimization to be carried out in real time [1, 2]. Hence there is a need for suitably modifying *CLO* techniques to have faster response time in comparison with the traditional Single Layer Optimization (*SLO*). Both *CLO* and *SLO* have been implemented in this paper by using a variant of Particle Swarm Optimization that uses Digital Pheromones referred to as *PSODP*. The experimental results presented in the paper indicate the speedup obtained by the parallel implementation of the *CLO* in comparison with the *SLO*.

Traditionally in layered network operating systems only adjacent layers have been communicating and fine tuning themselves to work in an optimized way [1]. But Cross Layer Designs have been exploring the possibility of communication between all the layers in order to work in an optimized manner. Cross Layer optimization can be integrated into the existing wired and wireless systems without fundamentally changing their original design. Centralized and decentralized schemes [4] have also been proposed to cater to network layers from the same and different manufacturers respectively.

Particle Swarm Optimization (*PSO*) [5, 6] is a computational method that optimizes a problem by iteratively trying to improve a candidate solution based on a fitness function. The solution characteristics could be improved by using digital pheromones in *PSO* [7]. *PSO* and *PSODP* are techniques inspired from the field of *swarm intelligence* where the interactions between swarm members require no supervision or prior knowledge and is based on primitive instincts. Domain specific real time problems are solved using particle swarm optimization like *PSO* for reactive power and voltage control [9] in electric power systems. The faster convergence of *PSO* has also been obtained in multidimensional problem space [10], and it has been used as an optimizer [11], with fuzzy logic [12] and for genetic algorithms [13]. The real time cross layer problem can be implemented both sequentially [14] as well as in a parallel mode [15]. This paper, adopts the parallel version of the Particle Swarm Optimization (*PPSO*) proposed in [8]. The utilization of *PSODP* for *CLO* and *SLO* is proposed and their parallel implementations are presented here. The performance evaluation and the analysis indicate that *CLO* has significant speed up in response time and faster convergence compared to *SLO*.

2. RELATED WORK AND MOTIVATION

The need for Cross Layer Optimization (*CLO*) in the existing wireless communication systems [1] has been presented in this section. Particle Swarm Optimization and its variant that uses digital pheromones [7] will also be briefly introduced here in order to highlight the motivation behind its use in our proposed work. The issues related to the parallel implementation of these optimization techniques are also discussed here after a brief introduction to Single Layer Optimization (*SLO*).

2.1 SINGLE LAYER OPTIMIZATION (SLO)

In Network Operating Systems (*NOS*), each layer in the traditional Open System Interconnection (*OSI*) model has been optimized individually. The algorithms and protocols in the various layers are designed to optimize themselves independently as they have different objectives. For instance in multimedia communication, various layers operate on different parameters related to the multimedia traffic and thereby take as input different types of information. Here the physical layer is concerned with the actual bits of information and depends heavily on the channel characteristics, while the application layer is concerned with the semantics and dependencies between flows and depends heavily on the multimedia content. Thus this analysis on *SLO* helps to propose a performance comparison between *SLO* and *CLO* for efficient utilization of network resources, as in *CLO* the strategies and parameters of the layers are optimized jointly.

2.2 CROSS LAYER OPTIMIZATION (CLO)

In Network Operating Systems, Cross Layer Optimizations (*CLO*) are performed with the objective of selecting a joint strategy across multiple layers. Normally the three layers taken into consideration for *CLO* are the Physical (*PHY)* layer, Medium Access Control (*MAC)* layer and the Application (*APP)* layer. The adaptation and protection strategies of the layers are represented as

(i) PHY_i, i $\in$ {1,2,….*NP*}

(ii) MAC_i, i $\in$ {1,2,….*NM*}

(iii) $APP_{i,}$, i $\in$ {1,2,….*NA*}

where *NP*,*NM*, *NA* represents the number of strategies at the respective layers. The strategies that can be considered for the above layers are for instance,

- Modulation
- Channel Coding Schemes

for the physical layer,

- Packetization
- Automatic Repeat Request (*ARQ*)
- Scheduling
- Admission Control
- Forward Error Correction (*FEC*) mechanisms

for the MAC layer and

- Video Compression parameters
- Packetization
- Traffic shaping
- Traffic Prioritization
- Scheduling
- *ARQ* and *FEC* mechanisms

for the application layer respectively.

The joint cross layer strategy *S* can be defined as, $S = \{PHY_1 \ldots PHY_{NP}, MAC_1 \ldots MAC_{NM}, APP_1 \ldots APP_{NA}\}$, that yields, $N = NP \times NM \times NA$ possible joint design strategies. The objective of the cross-layer optimization problem is to find the optimal joint strategy that results in the best utilization *U* as indicated in the Eq.(1),

$$S^{opt}(x) = max\ U(S(x)) \quad (1)$$

where *x* represents the constraints like bandwidth, packet loss ratio, delay, power, etc,. Cross Layer Optimization solutions are traditionally classified [2] into several categories and briefly presented below,

1. Top down Approach: The higher layer protocols optimize their parameters and strategies at the next lower layer.
2. Bottom up Approach: The lower layers try to insulate the higher layers from losses and bandwidth variations.
3. Application Centric Approach: The APP layer optimizes the lower layer parameters one at a time in a Bottom up or Top down manner based on its requirements.
4. MAC centric Approach: The APP layer passes its traffic information and requirements to the MAC which decides which APP layer packets should be transmitted and at what QoS level.
5. Integrated Approach: Here the strategies are determined jointly. The proposed work uses this approach in effectively performing the cross layer design to determine the optimal cross layer strategy.

2.3 PARTICLE SWARM OPTIMIZATION (PSO)

Particle Swarm optimization [5,6] is an emerging evolutionary computing technique that is simple and easy to implement and helps to achieve relatively faster convergence.

2.3.1 Basic Particle Swarm Optimization (BPSO):

In the Basic *PSO*, *P* numbers of particles are randomly distributed in a problem solution space *S* with *N* number of dimensions represented as S^N. Each particle will compute the solution and determine their suitability by using the fitness function $f(s^1,s^2,\ldots,s^n)$, where $0 < n \leq N$ and $s^n \in S^N$. The objective is to find a set of $S' \subset S$ to maximize/minimize the fitness function according to Eq.(2).

$$S' = argmax\ f\ (s^1, s^2, \ldots s^n) \quad (2)$$

The velocity and the position update which are the primary computations carried out in Basic *PSO* (*BPSO*) are given below,

$$v[\,] = v[\,] + c1*r()*\left(pb[\,]-pr[\,]\right) + c2*r()*\left(gb[\,]-pr[\,]\right) \quad (3)$$

$$pr[\,] = pr[\,] + v[\,] \quad (4)$$

v[] is the particle velocity, pr[] is the current particle position. pb[] represents the particle's best position and gb[] refers to the global best position. r()is a random number between (0,1). c1 and c2 are the acceleration coefficients (usually c1 = c2 = 2). The pseudo code of the Basic PSO is given below,

Algorithm : *BPSO ()*

For ***each particle*** *do*
Initialize particles
End For
While ***maximum iterations or minimum error criteria is not attained*** *do*
For ***each particle*** *do*
Calculate fitness value
If ***the fitness value is better than the best fitness value (pBest) in history*** *then*
set current value as the new pBest
End If
End For
Choose the particle with the best fitness value all the particles as the gBest
For ***each particle*** *do*
Calculate particle velocity according to Eq.(3)
Update particle position according to Eq.(4)
End For
End While

2.3.2 PSO with Digital Pheromones (PSODP):

A variant of *PSO* [7] uses Digital Pheromones for aiding communication within the swarm to improve the rate of convergence. A Digital Pheromone that decays after regular intervals of time is like a natural pheromone which is a chemical secreted by an animal, especially an insect that influences the

behavior or development of others of the same species. This additional Digital Pheromone component potentially causes a swarm member to result in a direction different from the combined influence of the *particle's best* and *global best* positions. This thereby increases the probability of finding the global optimum. The velocity update is done using the formulation given below,

$$v[] = v[] + c1 * r() * (pb[] - pr[]) + c2 * r() * (gb[] - pr[]) + c3 * r() * (Tph[] - pr[])$$

The parameter *c3* is the user defined *confidence* parameter for the pheromone component of the velocity vector. *c3* combines the knowledge from the *cognitive* and *social* components of the velocity of a particle and complements their deficiencies. The *confidence* parameter *c3* determines the extent of influence a target pheromone can have on the swarm when the information from *particle's best* and *global best* alone are not sufficient to determine a particle's next move. The particle is attracted to a target pheromone *Tph[]* that has the highest *P'* value based on its proximity to other pheromones and their pheromone level. *P'* is given by,

$$P' = (1 - d)\, P$$

where *d* is the distance between the particle and the target pheromone.

3. THE PROPOSED WORK

This section presents the proposed algorithms for Single-Layer Optimization (*SLO*) and Cross-Layer Optimization (*CLO*) using Particle Swarm Optimization with Digital Pheromones (*PSODP*). The theoretical and experimental analysis helps to evaluate the performance comparisons of *CLO* and *SLO*.

3.1 ALGORITHM FOR SINGLE LAYER OPTIMIZATION USING PSODP

The algorithm for Single Layer Optimization (*SLO*) using Particle Swarm Optimization with Digital Pheromones is given below, along with the description of the algorithm.

Algorithm 1: *SLO ()*

For ***each OSI layer considered*** *do*

1 *Determine all the strategies and parameters;*
2 *Define all the strategies and parameters in a lookup table;*
3 *Call PSODP();*
4 *Determine the best output(strategy/parameter);*
5 *Refer to the lookup table;*
6 *Return the corresponding strategy/parameter;*

End For

The *OSI* layers considered here are the Physical (*PHY*) layer, Medium Access Control (*MAC)* layer and the Application (*APP*) layer. For each layer considered for optimization all the strategies and parameters are determined and defined in a look up table. The Algorithm *PSODP ()* is invoked by each layer to determine the best output. Finally the look up table is referred to return the corresponding strategy/parameter. The fitness function used should be designed in such a way that it considers the previous layer's output as important parameters. Particle Swarm Optimization with Digital Pheromones (*PSODP*) [7] is efficiently utilized in this paper for the parallel implementation of *CLO*.

Algorithm 2: *PSODP ()*

Populate the swarm with random initial values

While ***!converged*** *do*

1 *Evaluate the fitness value of each swarm member*
2 *Store pbest and gbest*
3 *Decay digital pheromones in the solution space (if any)*
4 *If* ***Iteration==1*** *then*
Randomly choose 50 percentage of swarm to release pheromones
5 *Else*

Improved particles releases pheromones

6 *End If*
7 *Find target pheromone towards which each particle moves*
8 *Update velocity vector and position of each particle*

End While

Stop the algorithm

3.2 ALGORITHM FOR CROSS LAYER OPTIMIZATION USING PSODP

The algorithm for Cross Layer Optimization (*CLO*) using Particle Swarm Optimization with Digital Pheromones is given below along with the description of the algorithm.

Algorithm 3: *CLO ()*

1. For ***each OSI layer considered*** *do*

i) Determine all the strategies and parameters;
ii) Define all the strategies and parameters in a lookup table;

End For

2. Call PSODP ();
3. Determine the best combination of output;
4. For ***each layer considered*** *do*

i) Refer to the lookup table;
ii) Return the corresponding strategy;

End For

5. Determine the best joint strategy as output;

All the strategies and parameters are determined for each *OSI* layer considered and they are defined in a look up table. A call to *PSODP ()* is performed which determines the best output considering all the layers together. The look up table is referred to and the best joint strategy is returned.

The problem space considered here is three dimensional. This 3-D space is divided into many partitions by giving limits to Z-axis. Rather than determining the solution in the problem space sequentially, the parallel implementation of resolving *CLO* is designed. Here for every partition that runs in parallel, the

PSODP () is called to determine the best output based on a fitness function. The final joint strategy is determined by considering the output of every partition and is based on the objective of the fitness function.

3.3 MOTIVATION OF THE PROPOSED WORK

A theoretical comparison of the performance of Single Layer and Cross Layer Optimization is presented here to motivate on the understanding of our proposed work.

a) In *SLO*, layers are optimized individually and there is no provision for a feedback mechanism. It is only through this feedback mechanism that the lower layers communicate to the higher layers the discrepancies in selecting the optimal strategy and the need to change the higher layer strategies. The Cross Layer approach transports feedback dynamically via the layer boundaries to enable the compensation, for example overload, latency or other mismatch of requirements and resources by some control input.
b) While Cross-layer Optimization contributes to an improvement of quality of services under various operational conditions, the *SLO* strategy does not take into consideration the quality of service issues and hence does not result in any significant improvement in the network utilization. Rather than optimizing each layer individually, the *CLO* performs joint analysis, selection and adaptation of various combinations of strategies available at different layers. This leads to better utilization of power and spectrum of the network.

The above theoretical facts also enable to compare the performance of *SLO* and *CLO* experimentally in order to utilize the network resources efficiently.

4. RESULTS AND ANALYSIS

This section analyses the comparison between *SLO* and *CLO* both theoretically and experimentally.

4.1 THEORETICAL ANALYSIS

Since Cross-Layer Optimization is a joint optimization of transmission strategies across layers there is a potential for the parallel implementation of *CLO* using PSODP. But in *SLO*, optimizations are performed individually at each layer where the parallel method of resolving *SLO* takes the same time as that of sequential implementation. This analysis helps in understanding that the response time of resolving *CLO* is faster compared to the time of resolving *SLO*. For instance if it takes '*k*' time units to resolve *CLO* using *PSODP*, it takes '*3*k*' time units to resolve *SLO* using *PSODP* as the number of layers considered in our proposed work is three.

In *CLO*, the optimization is efficiently performed by jointly analyzing the layers and effectively using the feedback mechanism. But in *SLO*, there is no feedback provision and hence the result may not be optimal.

4.2 EXPERIMENTAL RESULTS AND ANALYSIS

This section describes the parallel implementation of the proposed work using Compute Cluster, the Message Passing Interface (*MPI*) and Boost Libraries after a brief introduction about the different types of clusters available. The result and analysis of the experiments carried out shows significant results supporting the claims that are made in the paper.

4.2.1 Types of Cluster:

The following are the different types of cluster systems used in real time applications,

a) ***High-availability (HA) clusters***
High-availability clusters (also known as Failover Clusters) are implemented primarily for the purpose of improving the availability of services that the cluster provides. They operate by having redundant nodes, which are then used to provide services when system components fail. The most common size for an HA cluster is two nodes, which is the minimum requirement to provide redundancy. HA cluster implementations attempt to use redundancy of cluster components to eliminate single points of failure.
b) ***Load-balancing clusters***
Load-balancing is performed when multiple computers are linked together to share computational workload or function as a single virtual computer. Logically, from the user side, they are multiple machines, but function as a single virtual machine. Requests initiated from the user are managed by, and distributed among, all the standalone computers to form a cluster. This results in balanced computational work among different machines, improving the performance of the cluster systems.
c) ***Compute clusters***
Often clusters are used primarily for computational purposes, rather than handling IO-oriented operations such as web services or databases. For instance, a cluster might support computational simulations of weather or vehicle crashes. The primary distinction within computer clusters is how tightly-coupled the individual nodes are. For instance, a single computer job may require frequent communication among nodes - this implies that the cluster shares a dedicated network, is densely located, and probably has homogenous nodes. This cluster design is usually referred to as Beowulf Cluster.

The other extreme is where a computer job uses one or few nodes, and needs little or no inter-node communication. This latter category is sometimes called Grid computing. Tightly-coupled compute clusters are designed for work that might traditionally have been called supercomputing. Middleware such as Message Passing Interface (MPI) or Parallel Virtual Machine (PVM) permits compute clustering programs to be portable to a wide variety of clusters.

4.2.2 Message Passing Interface (MPI) and Boost Libraries:

Message Passing Interface is an *API* specification that allows processes to communicate with one another by sending and receiving messages. It is typically used for parallel programs running on computer clusters and supercomputers, where the cost of accessing non-local memory is high. It is a language-independent communication protocol used to program parallel computers. Both point-to-point and collective communication are supported. Its main goals are high performance, scalability, and portability. The MPI interface is meant to provide essential virtual topology, synchronization, and communication functionality between a set of processes (that have been mapped to nodes/servers/computer instances) in a language-independent way, with language-specific syntax

(bindings), and a few language-specific features. *MPI* programs always work with processes. Typically, for maximum performance, each *CPU* (or core in a multi-core machine) will be assigned just a single process. This assignment happens at runtime through the agent that starts the MPI program, normally called *mpirun* or *mpiexec*.

The Boost *C++* Libraries are a collection of free libraries that extend the functionality of *C++*. They range from general-purpose libraries like the *smart-ptr* library, to operating system abstractions like Boost File System, to libraries primarily aimed at other library developers and advanced C++ users, like the template Meta Programming (*MPL*) and *DSL* creation (Proto). In order to ensure efficiency and extensibility, Boost makes extensive use of templates. Boost has been a source of extensive work into generic programming and Meta Programming in *C++*.

4.2.3 Parallel Implementation Using Compute Cluster:

In our proposed work the Compute cluster is used for the implementation of *CLO* and *SLO*. It is usually deployed to improve the performance and availability over that of a single processor, while typically being much more cost-effective than single processors of comparable speed or availability.

The system was programmed using *MPI* and Boost libraries in C++ programming language and run on a cluster system that consists of 8 nodes to evaluate the performance of *CLO* in comparison with *SLO*. Well known and widely used single objective fitness function called the Rastrigin function was used as presented in eq. (5).

4.2.4 Results and Analysis:

The experimental results were obtained using a computing platform as explained in the previous section. The parallel implementation of the proposed work was executed and a graph was constructed between the Clock Cycles (for convergence) in X-axis and the corresponding Fitness values (obtained using Rastrigin function) in the Y-axis as in Fig.1. This graph indicates the faster convergence of *CLO* using *PSODP* with few clock cycles compared to the implementation of *SLO* using *PSODP*. The output convergence is obtained using Rastrigin function as shown in Eq.(5).

$$f(x) = \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) + 10 \right) \quad (5)$$

The value of x_i ranges between [-5.12, 5.12]

where the parameter '*n*' represents the dimension of the problem space. In the proposed work the problem space is considered to be of dimension three representing the Physical (*PHY*), *MAC* and Application (*APP*) layers.

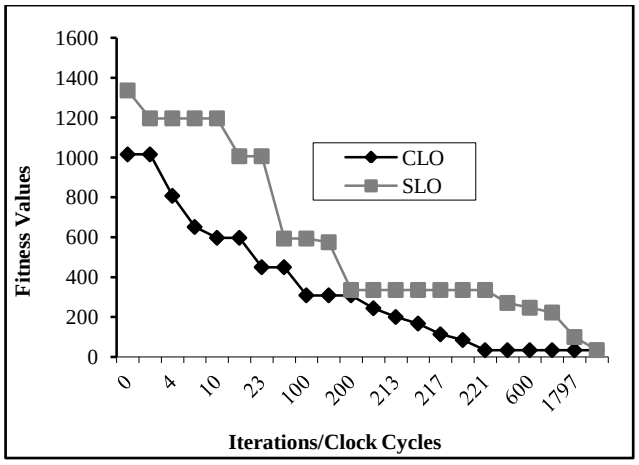

Fig.1. Convergence of Fitness Values

The graph in Fig.2 is constructed for every periodic interval of Time Units (X-axis) with the respective Computation Time (Y-axis) for both the approaches. This graph indicates that the computation time required to reach the global optimum using *PSODP* is very less in *CLO* compared to *SLO*.

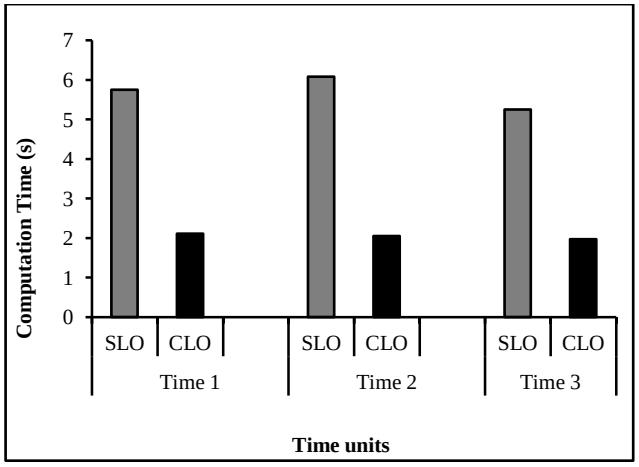

Fig.2. Comparison of Computation Time

5. CONCLUSION

This paper proposes to compare *SLO* and *CLO* using a variant of Particle Swarm Optimization with Digital Pheromones (*PSODP).* The main purpose of using *PSODP* is for faster convergence towards the global optimum. The experimental results presented show that the proposed parallel implementation of *CLO* has significantly faster response time in comparison with *SLO*.

REFERENCES

[1] M. van Der Schaar and N. Sai Shankar, "Cross-layer wireless multimedia transmission: challenges, principles, and new paradigms", *IEEE Wireless Communications, [see also IEEE Personal Communications]*, Vol. 12, No. 4, pp. 50-58, 2005.

[2] S. Ci, H. Wang, and D. Wu, "A theoretical framework for quality-aware cross-layer optimized wireless multimedia

communications", *Advances in Multimedia- Hindawi Publishing Corporation*, Vol. 2, No.1, pp. 2-10, 2008.

[3] S. Shakkottai and P. C. Karlsson, "Cross-layer design for wireless networks", *IEEE Communications Magazine*, Vol. 41, pp.74-80, 2003.

[4] N. Mastronarde and M. van der Schaar, "Online layered learning for cross-layer optimization of dynamic multimedia systems", *Proceedings of the first annual ACM SIGMM Conference on Multimedia systems, MMSys '10, New York, NY, USA* pp. 47-58, , 2010.

[5] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization. Swarm Intelligence", Vol. 1, pp. 33-57, 2007.

[6] D. Sedighizadeh and E. Masehian, "Particle swarm optimisation methods, taxonomy and applications", *International Journal of Computer Theory and Engineering*, Vol. 1, pp. 1793-8201, 2009.

[7] V. Kalivarapu, J.-L. Foo and E. Winer, "Improving solution characteristics of particle swarm optimization using digital pheromones", *Structural and Multidisciplinary Optimization- SpringerLink*, Vol. 37, No. 4, pp. 415-427, 2009.

[8] J.-F. Chang, S.-C. Chu, J. F. Roddick and J.-S. Pan, "A parallel particle swarm optimization algorithm with communication strategies", *Journal of Information Science and Engineering*, Vol. 21, No. 4, pp. 809-818, 2005.

[9] H. Yoshida, K. Kawata, Y. Fukuyama and Y. Nakanishi, "A particle swarm optimization for reactive power and voltage control considering voltage stability", *International Conference on Intelligent Systems Applications to Power Systems ISAP*, pp. 1-6, 2007.

[10] J. K. M. Clerc, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space", *IEEE Transactions on Evolutionary Computation*, Vol. 6, No.1, pp. 58-73, 2002.

[11] Shi Y and Eberhart R.C, "A modified particle swarm optimizer", *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 69-73, 1998.

[12] Shi Y and Eberhart R.C, "Fuzzy adaptive particle swarm optimization", *Proceedings of IEEE International Conference on Evolutionary Computation*, Vol. 1, pp. 101-106, 2001

[13] J. S. Pan, F. R. McInnes and M. A. Jack, "Application of parallel genetic algorithm and property of multiple global optima to vq codevector index assignment", *Electronics Letters*, Vol. 32, No. 4, pp. 296-297, 1996.

[14] Susan Elias, Vanaja Gokul, Kamala Krithivasan, Marian Gheorghe and Gexiang Zhang, "Real time Cross Layer Design using Particle Swarm Optimization", *Proceedings of the Fourth Annual Conference of ACM Bangalore, COMPUTE-2011*, pp. 25, 2011.

[15] Vanaja Gokul. R, Vigneshwaran. S, Thiagarajan. T. S and Susan Elias, "High Performance Optimization based on Swarm Intelligence", *International Conference on Emerging Technology Trends in Cloud Computing*, pp.1059-1064, 2011.

[16] Hsu, Cheng-Hsin and Hefeeda Mohamed, "A framework for cross-layer optimization of video streaming in wireless networks", *ACM Transactions on Multimedia Computing, Communications and Applications (TOMCCAP)*, Vol. 7, pp. 1-28, 2011.

[17] http://www.enotes.com/topic/Computer_cluster

[18] http://en.wikipedia.org/wiki