Adaptive Prediction in Desktop Grids

P J Gandhi ,N A Memon, S A Patel, S. Munot and Y V Chavan

Abstract-**Desktop grids are more popular platform for high throughput applications, but due to their inherent resource volatility it is difficult to exploit them for the applications that require rapid turnaround. Applications in a company's or institute's workload often require rapid turnaround. So desktop grids in an enterprise or an institute are often underutilized. Efficient resource management for short lived applications is required to implement desktop grids at institute or enterprise level. A novel idea of "Self managing resource selection for desktop grids based on adaptive prediction" is proposed in this paper. For efficient source selection 'online prediction of running time of application' is very much important concept. In this paper the concept of running time prediction is implemented using adaptive algorithms. To improve the coverage, span and confidence interval basic adaptive predictors as well as a new Adaptive estimator scheme is implemented. By implementing the adaptive schemes merely 100% coverage is obtained which is far better than the desirable 95% coverage compare to earlier schemes.**

Keywords: Grid Computing, Adaptive Predication, Resource selection in desktop Grid.

I. INTRODUCTION

Desktop grids use the idle cycles of mostly desktop PC's to support large-scale computation and data storage. Today, these types of computing platforms are the largest distributed computing systems in the world. The most popular project, SETI@home, uses over 20 TeraFlop/sec provided by hundreds of thousands of desktops. Indeed, it is a well-studied fact that machines primarily devoted to regular human-dependent interactions, like e-office applications (work processing, spreadsheets, etc.) and e-communications such as instant messaging, e-mail, and Internet browsing barely use their resources. For instance, Heap [5] reports nearly 95% CPU idleness amongst Unix machines, with an even higher value of 97% measured in Windows machines assigned to academic classrooms[14]. Furthermore, in their comprehensive study of more than 200000 SETI@home [19] hosts, Anderson and Fedak [6] report that an average of 89.9% of CPU was volunteered to public computing through the BOINC platform [7], meaning that roughly 90% of CPU would have been wasted if it was not exploited by BOINC projects. Numerous other projects, which span a wide range of scientific domains, also use the cumulative computing power offered by desktop grids, and there have also been commercial endeavors for harnessing the computing power within an *enterprise*, i.e., an organization's local area network.

Despite the popularity and success of many desktop grid projects, the volatility of the hosts within desktop grids has been poorly understood. Due to lack of resource management techniques, application of desktop grid is limited to high throughput applications that consist of very large number of tasks. But in an enterprise or institute most of the time applications consist of small or moderate number of tasks. In this case desktop grids are underutilized. So it's a big issue of research that how to use grid networks effectively for *short lived application.* In this paper we have tried to provide solution for this so that grid networks can be used in enterprises & institutes. The solution provided in this paper is basically based on the Adaptive prediction of resource availability.

So this paper leads us to a solution to develop a grid network in an enterprise or institute which can perform short lived as well as high throughput applications efficiently.

The rest of this paper is organized as follows. Section II describes the problem definition regarding the implementation of short lived applications efficiently. Section III describes various scheduling algorithms. Section IV presents the idea of adaptive application for resource management. Section V describes simulated prediction results in MATLAB. Section VI concludes the work.

II. IMPLEMENTATION OF SHORT LIVED APPLICATION ON DESKTOP GRID

A. *Grid functioning*

We consider the problem of scheduling an application that consists of T independent, identical tasks onto a desktop grid. The desktop grid comprises N hosts that can execute application tasks. These hosts are individually owned and can only be used for running application tasks when up and when their CPU is not used by their owners, making host and CPU availability dynamic. The hosts are managed by a master, which will call the "server", in the following way. The server holds all the input data for 'T' tasks, when any of the hosts gets free it sends notification to the server. The server maintains a queue of available hosts i.e. "Ready Queue" and may choose any one of them to complete the task. When owner of the host runs an application task, for time being process is suspended and it can be resumed on the same host later on. When an application task is running on the host, it

P J Gandhi ,N A Memon, S A Patel, S. Munot and Y V Chavan are with Maharashtra Academy of Engineering, Alandi (D), Pune

Gandhi.pallav@gmail.com, Noor.876@gmail.com, Suchit_gollu@yahoo.com, sachin.munot@gmail.com, yvchavan@maepune.com

sends "Heart Beats" to server every minute. If in between host is switched off then same task can be restarted on another host.

B. *Problem statement*

Given above problem model, we consider the case in which 'T' tasks are to be completed on 'N' hosts. For the applications which consist of very high number of tasks T >>N i.e. number of tasks is greater than number of hosts. In this application **FCFS (First Come First Serve)** gives the optimal performance. This is the strategy used by almost all the existing desktop grids like SETI@HOME,BOINC etc.

In this paper we focus on short live applications i.e. T<=N. For T=N is sub optimal and for T<N performance of FCFS is poor.

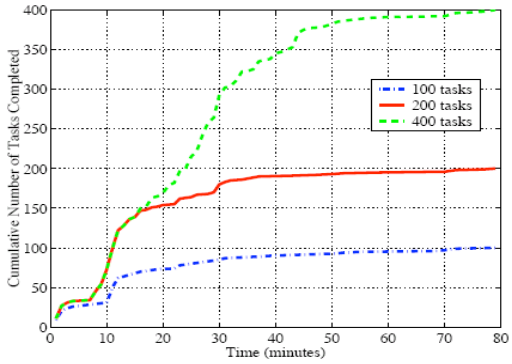

Figure-1 Task Completion vs. Time [9]

Above simulation results are obtained for N=190, 1.5 Ghz each and T = 100,200,400 tasks. As we can see for T = 100 (T<<N) performance is poor, T=200(T=~ N) performance is sub optimal and for T=400 (T>>N) performance is almost optimal.

So, from above results it is clear that for short live application there must be some resource selection scheme which can improve the performance.

III. SCHEDULING METHODOLOGIES

Traditional eager scheduling algorithms like First Come First Served (FCFS), which yield good performance in high-throughput oriented systems, are normally inefficient if applied unchanged in environments where fast turnaround times are sought. In order to adapt FCFS to fast turnaround-oriented environments several changes are applied. First, support for shared checkpoints are added. Under the shared model, checkpoints are kept in a common repository and not exclusively at the executing machine's storage. This permits checkpoints to be shared amongst machines, allowing tasks to be resumed, moved or replicated to another machines, accordingly to the circumstances. This effectively provides for task mobility, and consequently for an improved usage of resources relatively to private checkpointing. A requirement of the shared checkpoint model is the existence of a storage infrastructure accessible to all machines and able to support the possible concurrent access of the computers involved in computation. In a medium-sized institutional environment, a regular PC acting as file server should suffice to implement a shared checkpoint service. Furthermore, this service can be located jointly with the scheduler service, to promote synergies between the scheduling and the checkpointing systems. Based on shared checkpoints, and using FCFS as base, several scheduling policies were devised, namely FCFS-AT (AT stands for *adaptive timeout)*, FCFS-TR (*task replication*) and FCFS-PRDCT-DMD (*prediction on demand*).

A. *FCFS-AT*

FCFS-AT improves the base FCFS with the inclusion of an adaptive timeout that sets a time limit for the termination of a task. This timeout defines the maximum time conceded to the machine for completing the execution of the assigned task. Should the timeout expire before the task has been completed, the scheduler considers the task as non-completed and can therefore reassign it to another machine, where it will be restarted, possibly from the last stable checkpoint (assuming shared checkpoint is enabled).

The timeout is defined when a task is assigned to a requesting machine and its duration takes into account the needed CPU reference time to complete the task as well as the machine computing performance as given by the Bytemark benchmark indexes [4]. These values are used to compute the minimum time needed by the candidate machine to complete the task, estimating an ideal execution, that is, a fully dedicated and flawless machine. To accommodate for reality, a tolerance is added to the base timeout. This tolerance, defined by way of a percentage of the timeout's base-time, varies accordingly to the CPU reference time still needed to complete the task (the bigger the reference CPU time needs, the bigger the tolerance percentage). However, if the execution is scheduled for a night-period or for a weekend, the tolerance is fixed, respectively to 10% and 5%. This is to take advantage of the stability of the desktop grid resources during period of low or non-existence of human presence.

B. *FCFS-TR*

FCFS-TR adds task replication on top of the FCFS-AT scheduling policy. The main strategy is to resort to task replication at the terminal phase of the computation, when all uncompleted tasks are already assigned and there is at least one free machine. The underlying principle is that replicating a task, especially if the replica is scheduled to a faster machine than the current one, augments the probability of a faster completion of the task, and thus might reduce the turnaround time. Even, if the replica is assigned to a slower or equal performance machine, it can still be useful acting as backup in case if the primary machine fails or get delayed.

In order to avoid excessive replicas of a same task, something that could perturb the balanced access to resources, the number of replicas of a task is maintained under a predefined threshold. Therefore, tasks which level of replication has already reached the limit can only be further replicated when one of the current replicas is interrupted. Furthermore, when a task is terminated all other results produced by replicas that might exist are simply discarded (we do not consider redundancy for sabotage-tolerance purposes, assuming that all computing resources behaves honestly).

C. FCFS-PRDCT-DMD

The FCFS-PRDCT-DMD scheduling policy resorts to short-term prediction regarding machines' availability on top of FCFS-TR. When a prediction indicates that a currently requested machine might fail in the next scheduling interval, the scheduler orders a checkpoint (henceforth referred as "checkpoint on demand") and then promotes the creation of a replica if conditions are met (that is, at least a free machine exists and the maximum number of replicas for the considered task has not yet been reached). The rationale behind this policy is to anticipate unavailability of machines, taking the proper measures to reduce or even eliminate the effect of the machine unavailability on the execution of the application. In this work, the prediction method used was the sequential minimal optimization (SMO) algorithm which yielded the best prediction results in a previous study [1].

Here in this paper we will define prediction of running time of task using Adaptive LMS Estimator. To understand that first of all we will see the adaptive as well LMS Estimator.

IV. ADAPTIVE ESTIMATION TO MAKE ONLINE PREDICTION OF RUNNING TIME OF TASK

A. Online Prediction of Running time of Applications.

To provide consistent high performance when running on typical shared, unreserved distributed computing environments, adaptive applications must exploit the degrees of freedom such environments offer, carefully choosing how and where to run their tasks. To make such decisions, applications require predictions of the performance of each of the alternatives. If the application could predict the running time of the task on each of the available hosts, it could trivially choose an appropriate host to run the task. Even if no host existed on which the task could meet its original deadline, such predictions of running time would permit the application to modify the resource requirements of the task or its deadline until an appropriate host could be found.

Evaluating the quality of the confidence interval [*tlb,tub*] is a somewhat complex endeavor. Suppose we ran a wide variety of testcases with a specified confidence, say 95%. If we used the ideal algorithm for computing confidence intervals and the best possible predictor, the lengths of the tasks' confidence intervals would be the minimum possible such that 95% of the tasks would have running times in their predicted intervals. An imperfect algorithm, such as ours, will compute confidence intervals that were larger or smaller than ideal where fewer or more than 95% of the tasks complete in their intervals. The important point is that to evaluate a confidence interval algorithm, we must measure the lengths of the confidence intervals it produces and the number of tasks which complete within these confidence intervals. To evaluate confidence intervals, we will use following two metrics:

Coverage: The fraction of tasks which complete with their predicted confidence intervals.

Span : The average width of the confidence interval width in seconds.

The ideal system will have the minimum possible span such that the coverage is more than 95%.

Now in this paper prediction method which is used is Adaptive LMS Estimator. To understand LMS Estimator we will first go through Adaptive Estimator first in section 4.2 and than LMS Estimator in section 4.3.

B. The Adaptive Estimator [3]

Consider the estimator as an FIR structure with coefficients h(.), so that the (discrete) error at instant 'n',

$$e(n) = d(n) - \underline{h}^T(n)\,\underline{x}(n) \tag{4.1}$$

$$E\{e^2(n)\} = E\{d^2(n)\} + \underline{h}^T(n)E\{\underline{x}(n)\underline{x}^T(n)\}\underline{h}(n) - 2E[d(n)\underline{h}^T(n)\underline{x}(n)] \tag{4.2}$$

Defining $E\{\underline{x}(n)\underline{x}^T(n)\} = [R_{xx}(n)]$ as the autocorrelation matrix of input vector $\underline{x}(n)$, and $R^T{}_{dx} = E\{d(n)\underline{x}^T(n)\}$ as the cross correlation (vector) between d(n) and $\underline{x}(n)$, with $\underline{h}^T(n)\underline{x}(n) = \underline{h}(n)\underline{x}^T(n)$, one obtains mean squared error

$$\xi(n) = \sigma_{d^2} + \underline{h}^T(n)[R_{xx}(n)]\underline{h}(n) - 2\underline{R}_{dx}{}^T(n)\underline{h}(n) \tag{4.3}$$

Therefore minimum ξ(n) occurs for $\nabla_h\xi(n) = \frac{\partial\xi(n)}{\partial\underline{h}} = 0$, resulting in $\underline{h}_0(n)$ or

$$\underline{h}_{opt}(n) = [R_{xx}(n)]^{-1}[\underline{R}_{dx}(n)]^T \tag{4.4}$$

The continuous form of equation (4.4) is $\underline{h}_0(t) \otimes [R_{xx}(t)] = \underline{R}_{dx}(t)$, which is the celebrated (continuous) Wiener-Hopf integral equation whose solution (in principle) provides the optimum processor (coefficient) vector $\underline{h}_0(t)$ for stationary random processes $\underline{x}(t)$. The general solution to equation (4.4) has not been found till now, in continuous time. The matrix formulation of this desired equation is described in alternate fashion. The convolution operation $h(n) \otimes x(n)$ and the vector multiplication or inner product operation described above are essentially the same, in the context of the final result.

There are essentially two types of discrete solutions to equation (4.4) exemplified by the RLS (Recursive Least Squares) and the LMS (Least Mean Square) adaptive algorithms accepted as standard terminology, and briefly discussed in the sequel.

C. The LMS Estimator [18]

The RLS algorithm uses the least squares concept, which briefly denotes the implementation of minimizing the sum of squares of the difference between a time dependent function $f(t_i)$ and a relevant set of weighted measurements x_i according to an optimum fit requirement,

$$\sum \left\| \left[f(t_i) - \sum_{k=0}^{M-1} h_k x(i-k) \right] \right\|^2 = \text{Minimum}$$

where the h's are suitable weights/coefficients. One is following equations (4.1) to (4.4) in section. 4.1, where in $f(t_i) \equiv d(.)$ averaging summation and the weighted sum above is the equivalent of $\underline{h}^T(.)\underline{x}(.)$ of equation (4.1) in section 4.1. The LMS algorithm is a simplification of the RLS algorithm, wherein $\underline{g}(n) = [P(n)]\underline{x}(n)$ is replaced by $\mu \underline{x}(n)$ with μ a constant called the adaptation constant or step size parameter like $[P(n)]$ in the relevant literature. This is interpreted to mean that the statistically derived discrete parameter $[P(n)]$ is replaced by an instantaneous constant estimate of this parameter.

For an FIR (transversal type) processor with tap weights h(n) at instant 'n', k=0,1….M signifying the taps as exemplified , If $\xi(n) = E[d(n) - \sum_{k=0}^{M} h(k)x(n-k)]^2$ is the measured (squared) error with $h(k) = h_k(n)$, then the counterpart here of the weight update procedure adopted in the RLS scheme, follows the steepest descent concept. To apply this concept 'ξ' is considered related h(k) in inverted parabolic fashion, ξ is h(k). The objective is to find that h(k) recursively that occurs at the minimum of ξ. For this, starting at any $h_k(n)$ (for a given k), the time update $h_k(n+1)$ is corrected or adjusted with respect to $h_k(n)$ by a constant μ times the slope or gradient $\nabla_k(n)$, taken in the direction seeking the pit of the parabola.

Hence $h(n+1) = h(n) + \frac{\mu}{2}\{-\nabla_k(n)\}$, the factor ½ will be justified below. In this case $\nabla_k = \frac{\partial}{\partial h}\xi = 2E\{e(n)x(n-k)\} = -2R_{ex}(k)$

Where $R_{ex}(k)$ is the cross correlation between error e(n) and (delayed) tap input x(n-k).

In effect,

$$h(n+1) = h(n) + \mu R_{ex}(k) \tag{4.5}$$

In the LMS algorithmic approximation, $R_{ex}(.)$ is replaced by its instantaneous value i.e. error times tap input vector $e(n)x(n-k)$

$$\therefore h(n+1) = h(n) + \mu \underline{e}(n)\underline{x}(n-k) \tag{4.6}$$

Equation (4.6) is the LMS weight update equation; clearly the approximation of replacing the actual curve between ξ (n+1) and ξ(n) by the slope between these two values, will be better the smaller the μ(in principle). In practice because of the random nature of $\underline{x}(.)$ and e(.), this principle suffers consider blurring.

V. RESULTS

As it is mentioned in section 4.1 coverage and span these two are important parameters in online prediction of task. *The idle system should have coverage of 95% with minimum span.*

Here we have defined span in terms of accuracy in prediction. If our prediction is having percentage mean error performance of 1% only than we can define span

Average confidence Interval = Span = [tlb,tub]

where ***tlb*** *= predicted time – [percentage mean error . predicted time]*

tub *= predicted time + [percentage mean error . predicted time]*

Coverage is the fraction of tasks which complete with their predicted confidence intervals. So we can also define coverage in terms of percentage mean error. In fact

If our normal predictor is having percentage mean error of 1-2 % and we define the span as mentioned above than its' coverage will be merely 100%. This is the most desirable condition in online prediction of running time of task.

Here we have tried to achieve merely 100% coverage using LMS Estimator. For simulation of LMS Estimator, we have used a database shown in figure 2. Figure 3 shows the error performance of LMS Estimator. From the results it is clear that after convergence error in prediction is very low almost less than 0.5 %, where as mean of the percentage error is only 0.2442 % . So, using LMS Estimator we can predict he running time of task with less than 1% percentage prediction error . Now if we define confidence interval as above than coverage will be almost 100%. For the given case length of the confidence interval is only 0.4884% of the running time. If we provide the calculated confidence interval than more than 99% of the tasks will be having their running time lying into the confidence interval. So, coverage is more than 99% with very small confidence interval. Small Interval and more than 99% coverage is the most desirable feature to select resources most efficiently in desktop grid networks.

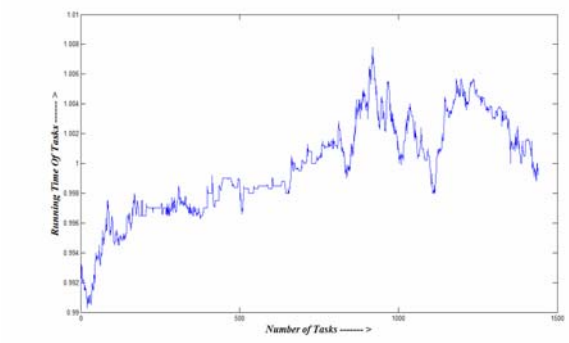

Figure 2. Database of almost 1500 tasks with their running time.

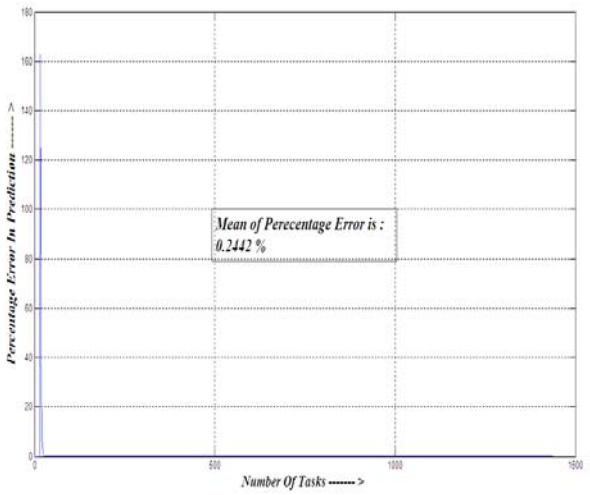

Figure 3. Error Performance Of LMS Estimator

VI. CONCLUSION

Thus in this paper we provided the solutions to implement multi purpose grids in enterprises as well in institutes. Desktop Grids are more famous for the applications with very large numbers of tasks. But Desktop Grids have failed to effectively deliver the results when used for small applications. Here in this paper we have proposed a self-managing resource selection scheme based on adaptive prediction which would help Desktop Grids to deliver better results when used for small applications and the data streaming capability will help the small grids to combine and deliver results for larger applications. The important concept of online prediction of running time of task is implemented using Adaptive Schemes. As results show the success to get merely 100% coverage is achieved using adaptive schemes.This will be a very useful concept for applications like audio-video streaming where coverage must be of minimum 95%.

VII. REFERENCES

[1] A. Andrzejak, P. Domingues, and L. Silva, "Classifier-based Capacity Prediction for Desktop Grids," presented at *Integrated research in Grid Computing - CoreGRID workshop*, Pisa,Italy, 2005.

[2] Berman, F. and R. Wolski: 1996, 'Scheduling From the Perspective of the Application'.In: *Proceedings of the Fifth IEEE Symposium on High Performance Distributed Computing HPDC96*. pp. 100–111.

[3] Bernard Widrow and Samuel D. Stearns, *Adaptive Signal Processing*, Pearson Education, 2004.

[4] BYTE, "BYTEmark project page (http://www.byte.com/bmark/bmark.htm)," Byte, 1996.

[5] D. G. Heap, "Taurus - A Taxonomy of Actual Utilization of Real UNIX and Windows Servers," IBM White Paper GM12-0191, 2003.

[6] D. Anderson and G. Fedak, "The Computational and Storage Potential of Volunteer Computing," 2005.

[7] D. Anderson, "BOINC: A System for Public-Resource Computing and Storage," presented at 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA., 2004.

[8] "Distributed Computing Info (http://distributedcomputing.info/)," 2006.

[9] D. Kondo, A. Chien, and H. Casanova, "Resource management for rapid application turnaround on entreprise desktop grids," presented at 2004 ACM/IEEE conference on Supercomputing, 2004.

[10] Dinda, P., B. Lowekamp, L. Kallivokas, and D. O'Hallaron: 1999, 'The Case forPrediction-Based Best-Effort Real-Time Systems'. In: *Proc. of the 7th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 1999)*, Vol. 1586 of *Lecture Notes in Computer Science*. San Juan, PR: Springer-Verlag, pp. 309–318. Extended version as CMU Technical Report CMU-CS-TR-98-174.

[11] D.Hanselman and B.Littlefield, *Mastering MATLAB 7*, Pearson Education, 2005.

[12] E.C.Ifeachor and B.W.Jervis, *Digital Signal Processing-A Practical Approach*, Pearson Education, 2005.

[13] Monson.H.Hayes, *Statistical Digital Signal Processing and Modeling*, John Wiley Publication, 2001.

[14] P. Domingues, P. Marques, and L. Silva, "Resource Usage of Windows Computer Laboratories," presented at International Conference Parallel Processing (ICPP 2005)/Workshop PEN-PCGCS, Oslo,Norway, 2005.

[15] Richard Bronson, *Theory and Problems of Matrix Operations*, Schaum's Outline Series, Mc Graw Hill Publications, 1988.

[16] Rudra Pratap, *Getting Started with MATLAB 7*, Oxford University Press, Indian edition, 2005.

[17] Simon Haykin, *Adaptive Filter Theory*, Pearson Education, 4th edition, 2002.

[18] Simon Haykin, *Digital Communications*, John Wiley Publication, 2001.

[19] SETI, "SETI@Home Project (http://setiathome.berkeley.edu/)," 2005.

VIII. APPENDIX

A. Database Used

Section A.1.1 is the small spread database which has been mainly used for the implementation of the various algorithms.

1) Small Spread Database Used

dataSS = x = [1.6280 1.6282 1.6280 1.6275 1.6270 1.6272 1.6272 1.6270 1.6270 1.6267 1.6265 1.6265 1.6268 1.6265 1.6266 1.6264 1.6260 1.6260 1.6258 1.6253 1.6258 1.6261 1.6260 1.6256 1.6256 1.6260 1.6260 1.6262 1.6258 1.6258 1.6255 1.6256 1.6262 1.6267 1.6270 1.6269 1.6265 1.6267 1.6265 1.6270 1.6275 1.6275 1.6270 1.6280 1.6285 1.6285 1.6290 1.6285 1.6287 1.6288 1.6288 1.6287 1.6287 1.6285 1.6287 1.6287 1.6295 1.6286 1.6285 1.6281 1.6283 1.6286 1.6288 1.6290 1.6265 1.6267 1.6272 1.6268 1.6260 1.6264 1.6270 1.6265 1.6265 1.6262 1.6265 1.6260 1.6265 1.6258 1.6258 1.6260 1.6257 1.6249 1.6248 1.6245 1.6249 1.6245 1.6250 1.6258 1.6253 1.6252 1.6255 1.6257 1.6248 1.6255 1.6250 1.6251 1.6248 1.6248 1.6259 1.6263 1.6263 1.6260 1.6262 1.6260 1.62654 1.6265 1.6265 1.6265 1.6265 1.6270 1.6275 1.6280 1.6279 1.6275 1.6280 1.6285 1.6285 1.6280 1.6280 1.6263 1.6280 1.6281 1.6287 1.6278 1.6278 1.6280 1.6283 1.6280 1.6283 1.6286 1.6265 1.6273 1.6276 1.6272 1.6266 1.6268 1.6271 1.6269 1.6267 1.6266 1.6266 1.6263 1.6265 1.6263 1.6263 1.6264 1.6260 1.6255 1.6256 1.6252 1.6251 1.6253 1.6257 1.6259 1.6255 1.6254 1.6258 1.6259 1.6254 1.6256 1.6253 1.6253 1.6253 1.6253 1.6255 1.6262 1.6267 1.6265 1.6263 1.6265 1.6264 1.6267 1.6270 1.6270 1.6268 1.6275 1.6280 1.6283 1.6285 1.6281 1.6284 1.6287 1.6286 1.6283 1.6284 1.6279 1.6283 1.6285 1.6290 1.6283 1.6282 1.6281 1.6283];

IX. ACKNOWLEDGEMENT

This paper is based upon the research work supported by Signal Processing Society of MAE-IEEE Student Branch.