ISSN:2231-0711
www.ijcset.net

A Survey and Taxonomy of scheduling algorithms in Desktop Grid

Jayoti Bansal
Computer Sc. & Engg., B.F.C.E.T, Bathinda Punjab (India),
erjyoti.2009@rediffmail.com

Dr. Shaveta
Computer Sc. & Engg., G.Z.S.C.E.T, Bathinda Punjab (India)
garg_shavy@yahoo.com

Dr. Paramjit Singh
Computer Sc. & Engg., B.F.C.E.T, Bathinda Punjab (India)
param2009@gmail.com

***Abstract*— Desktop Grids have proved to be a suitable platform for the execution of Bag-of-Tasks applications but, being characterized by a high resource volatility, require the availability of scheduling techniques able to effectively deal with resource failures and/or unplanned periods of unavailability. Scheduling BoT applications on Desktop Grids has thus attracted the attention of the scientific community, and various schedulers tailored towards them have been proposed in the literature However, due to the wide variety of approaches to this problem, it is difficult to meaningfully compare different systems since there is no uniform means for qualitatively or quantitatively evaluating them. It is also difficult to successfully build upon existing work or identify areas worthy of additional effort without some understanding of the relationships between past efforts. In this paper, taxonomy of approaches to the resource management problem in Grid environments is presented in order to provide a common terminology and classification mechanism necessary in addressing this problem.**

***Keywords*— Bag-of-Tasks applications (BoT), Desktop Grids, Scheduling taxonomy.**

I. Introduction

The exploding popularity of the Internet has created a new much large scale opportunity for Grid computing. As a matter of fact, millions of desktop PCs, whose idle cycles can be exploited to run Grid applications, are connected to wide-area networks both in the enterprise and in the home. These new platforms for high throughput applications are called Desktop Grids [1, 2], and provide an amount of raw computing power far exceeding that provided by more traditional Grid platforms that include a lower number of more powerful resources (e.g., high-performance clusters). The inherent wide distribution, heterogeneity, and dynamism of Desktop Grids make them better suited to the execution of loosely-coupled parallel applications rather than tightlycoupled ones. Bag-of-Tasks applications (BoT) [3, 4] (parallel applications whose tasks are completely independent from one another) have been shown [5] to be particularly able to exploit the computing power provided by Desktop Grids

In order to take advantage of Desktop Grid environments, a variety of widely differing techniques and methodologies for distributed resource management, tailored to BoT applications, must be adopted Along with these competing proposals has came the inevitable proliferation of inconsistent and even contradictory terminology, as well as a number of slightly differing problem formulations, assumptions, etc. Thus, it is difficult to analyze the relative merits of alternative schemes in a meaningful fashion. It is also difficult to focus common effort on approaches and areas of study which seem most likely to prove fruitful

The rest of the paper is organized as follows. Section 2 describing taxonomy in order to allows the classification of distributed scheduling algorithms according to a reasonably small set of salient features, while in Section 3 examples will be taken from the scientific literature to demonstrate their relationship to one another with respect to the taxonomy detailed in previous section. In Section 4 we will attempt to compare the scheduling algorithms previously described based on the taxonomy proposed Finally, Section 5 concludes the paper .

II. Taxonomy

The classification scheme proposes in our taxonomy considers six different scheduling categories represented in Figure 1: information-based and information-free, on-line and batch mode scheduling, fault-tolerant and not fault-tolerant.

A. *Information-based and Information-free scheduling*

The decisions a scheduler makes are only as good as the information provided to it. Many theoretical schedulers assume one has 100 percent of the information needed, at an extremely fine level of detail, and that the information is always correct. Unfortunately, as discuss later, this scenario is overly unrealistic. In general we have only the highest level of information. For example, it may be known that an application needs to run on Linux, will produce output files somewhere between 20 MB and 30 MB, and should take less than three hours but might take as long as five.

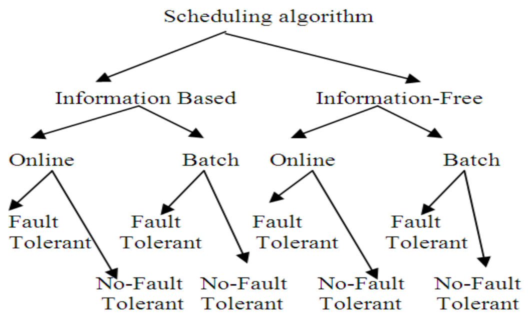

Figure1: Taxonomy

Usually, the information regards the tasks (execution time, software and hardware requirements, etc) and the resources (computational power, availability, etc). In general, the scheduling algorithms assume partial information, and use it to calculate a sort of utility value for each assignment task/resource. With this value the scheduler is able to evaluate how good is each assignment and decides how make the scheduling decisions. In some case, the goodness of an assignment could be decided also considering the cost to use a particular resource. As matter of fact, if the resources are freely usable any user could have an antisocial behavior (i.e., a single user can occupy the whole Computation Grid with his tasks). To avoid this situation, the scientific community has proposed some approaches based on microeconomy theories.

B. On-line and batch-mode scheduling algorithms

Independently of the amount of information about resources and task knew, the scheduling policies can be grouped into two categories: on-line and batch mode. In the on-line mode, a task is assigned to a machine as soon as it arrives at the scheduler and this decision is not changed once it is computed. Conversely, in the batch mode, tasks are not mapped onto the machines as they arrive; instead they are collected into a set that is examined for mapping at prescheduled times called mapping events. In the next section, we provide some example to put in evidenced the differences between on-line mode and batch mode scheduling.

C. Fault-tolerant scheduling

A Grid may potentially encompass thousands of resources, services, and applications that need to interact in order for each of them to carry out its task. The extreme heterogeneity of these elements gives rise to many failure possibilities, including not only independent failures of each resource, but also those resulting from interactions among them. Moreover, resources may be disconnected from a Grid because of machine hardware and/or software failures or reboots, network misbehaviors, or process suspension/abortion in remote machines to prioritize local computations. Finally, configuration problems or middleware bugs may easily make an application fail even if the resources or services it uses remain available [6]. In order to hide the occurrence of faults, or the sudden unavailability of resources, fault-tolerance mechanisms (e.g., replication or checkpointing-and-restart) are usually employed.

D. On-line mode scheduling

In the on-line mode scheduling, the scheduler must select the best resource given a particular task. The resource selection is done ordering the hosts in the ready queue according to some criteria (e.g., by clock rate, by the number of cycles delivered in the past) and to assign tasks to the "best" hosts first. This method is also called resource prioritization. The simplest on-line scheduling policy is First-Come-First-Serve (FCFS) where the scheduler selects the first host in the ready queue. Although this policy is not accurate since it does not consider any kind of information concerning either tasks or hosts, FCFS is often used also in important Desktop Grid projects like XtremWeb and BOINC. This is due to the fact that, in a dynamic and extremely heterogeneous environment like Grid it is difficult to obtain such information

III. Examples

In this section, examples will be taken from the scientific literature to demonstrate their relationship to one another with respect to the taxonomy detailed in previous section.

*A. On-**line mode scheduling***

In the on-line mode scheduling, the scheduler must select the best resource given a particular task. The resource selection is done ordering the hosts in the ready queue according to some criteria (e.g., by clock rate, by the number of cycles delivered in the past) and to assign tasks to the "best" hosts first. This method is also called resource prioritization.

The simplest on-line scheduling policy is First-Come-First-Serve (FCFS) where the scheduler selects the first host in the ready queue. Although this policy is not accurate since it does not consider any kind of information concerning either tasks or hosts, FCFS is often used also in important Desktop Grid projects like XtremWeb and BOINC. This is due to the fact that, in a dynamic and extremely heterogeneous environment like Grid it is difficult to obtain such information. Moreover many works, as described in [7], considers that when the number of tasks is about equal or greater than the number of hosts, there is just a little benefit of prioritization over FCFS. This can be explained considers that the most capable hosts tended to request tasks the most often, and so FCFS performed almost as well as any of the prioritization heuristic proposed in the scientific literature. Many of these scheduling policies are based on the classical on-line scheduling algorithms: minimum completion time (MCT), minimum execution time (MET), switching algorithm (SA), k-percent best (KPB) and opportunistic load balancing (OLB) (all described in detail in [8]).

For example, the work in [7] describes three methods for resource prioritization using different levels of information about the hosts from virtually no information to comprehensive historical statistics derived from trace for each host. In particular, for the PRI-CR method, hosts in the server's ready queue are prioritized by their clock rates. Similar to PRI-CR, PRICR- WAIT sorts hosts by clock rates, but the scheduler waits for affixed period of 10 minutes before assigning tasks to hosts. The rationale is that collection a pool of ready hosts before making the assignments can improve host selection. Finally, the third method called PRI-HISTORY uses dynamic information, i.e. history of a host's past performance to predict its future performance. All the results report in [7] are based on scenarios where the number of tasks is comparable with the number of resources. We believe that different scenarios where the number of tasks can be greater than the number of machines should be evaluated.

The work described in [9] is the most relevant in terms of Desktop Grid scheduling. The author investigates the problem of scheduling multiple independent compute bound applications that have soft-deadline constraints on the Condor Desktop Grid system. Each "application" in this study consists of a single task. The issue addressed in the paper is how to prioritize multiple applications having soft deadlines so that the highest number of deadlines can be met. The author uses two approaches. One approach is to schedule the application with the closest deadline first. Another approach is to determine whether the task will complete by the deadline using a history of host availability from the previous day, and then to randomly choose a task that is predicted to complete by the deadline. The author finds that a combined approach of scheduling the task that is expected to complete with the closest deadline is the best method. Although the platform model in that study considers shared and volatile hosts, the platform model assumes that the hosts have identical clock rates and that the platform supports check-pointing. So, the study did not determine impact of relatively slow hosts or task failures on execution for a set of tasks; likewise, the author did not study the effect of resource prioritization (e.g., according to clock rates) or resource exclusion.

B. Batch mode scheduling

In the batch mode scheduling, the scheduling approaches are typically more complicated with respect to the previous situation. As a matter of fact, the scheduler, besides the set of resources, knows the set of tasks and it can consider any combination of task/resource. Unfortunately, considering all possible combinations in order to decide the best set of assignments is a well-known NP-complete problem if throughput is the optimization criterion [10, 11, 12]. For this reason, the batch mode scheduling algorithms proposed in the scientific literature provides suboptimal solutions such as Min-Min, Max-min and Sufferage (all described in detail in [13]).

The most relevant batch mode scheduling policy is based on the classic algorithm cited above. For example, XSufferage [14] is an extension of Sufferage policy that is able to exploit file locality issues without any a priori analysis of the task-file dependence pattern. The idea is that if a file required by some task is already present at a remote cluster, that task would "suffer" if not assigned to a host in that cluster. The Sufferage's value would then be a simple way of capturing such situations and ensuring maximum file re-use. This is somewhat reminiscent of the idea of task/host affinities introduced in [15], where some hosts are better for some tasks but not for others. The problem of this algorithm regards the necessity to know much information to calculate the completion time of the tasks. In particular, it needs to know the HostSpeed (a measure of the host speed), the HostLoad (the load of the host due to the local processes) and the TaskSize (the completion time of a task in a host with HostSpeed=1 and HostLoad=0).

Moreover, as we have cited above, some scheduling algorithm needs an extremely fine level of detail such as First-order Prediction-based Dynamic FPLTF (FP Dynamic FPLTF) [16], abbreviated as FP. FP is a prediction-based scheduling algorithm that works as FPLTF [17] except that it needs the host's latest two load records. The scheduler uses these two records to reconstruct an approximated hosts loading model to predict the hosts's speed in the future based on the linear function. FP is able to achieve good performances but, it needs a level of detail about information of tasks and hosts that is overlay unrealistic to obtain in a Computational Grid, for the reasons previously discussed.

C. Fault-tolerant scheduling

The scheduling approaches discussed above do not consider the occurrence of faults, but as we have just observed, Grid environments are prone of failures. Task failures near the end of the application, and unpredictably slow hosts can cause major delays in application execution. Many solutions propose the replication of the tasks on multiple hosts, either to reduce the probability of task failure or to schedule the application on a faster host.

Replication a task may increase the chance that at least one task instance will be completed. One of these proposals is WorkQueue with Replication (WQR) [17]: a knowledge-free scheduling algorithm that adds task replication to the classical Workqueue (WQ) [17] scheduler. The beginning of the algorithm execution is the same as the simple Workqueue and continues the same until the bag of tasks becomes empty. At this time, in the simple Workqueue, hosts that finish their tasks would become idle during the rest of the application execution. Using the replication approach, these hosts are assigned to execute replicas of tasks that are still running. Tasks are replicated until a predefined maximum number of replicas are achieved. When a task replica finishes, its other replicas are canceled. This policy has the drawback of wasting CPU cycles (due to the replicas that do not contribute to the completion of the tasks), which could be a problem if the Desktop Grid is to be used by more than one application.

Conversely of these information-free schedulers, the scientific literature has proposed fault-tolerant schedulers that combine replication and information-based scheduling. For example, Distributed Fault-Tolerant Scheduling (DFTS) [18] is an on-line scheduling policy that uses job replication strategy and it considers available sufficient information to estimate the run-time of the task. In particular, when a job arrives, DFTS chooses a set of n candidate's sites for job execution and orders them for an estimate of the job completion time. If the desirable replication threshold is greater than n, DFTS reserves a set of resources equal to the number of unscheduled job replicas. If a job successfully completes, DFTS sends releases message to each site it had reserved such that these sites can be used for running other jobs. The experimental results of DFTS show that performances degrade gracefully in the presence of failures, but it does not consider the amount of waste cycles due to the useless replicas.

Besides the completion time and the number of the replicas desired, other scheduling policies specified also a minimum replication threshold. For example, in the Fault Tolerant Scheduling algorithm (FTSA) [19], along with the job, two values are specified by the user: the number of desired replicas n and the replication threshold k, where k <=n. FTSA picks the n best hosts (ordered by an estimate of the job completion time) and send them the replicas of the task. The system must ensure that k replicas are running. That is, the number of replicas may fall below n due to replica failure, but not below k. Although FTSA could achieve good performance even in presence of fault, it is not clear explained how the user should specify the values of n and k.

Finally, task checkpointing is another means of dealing with task failures since the task state can be stored periodically either on the local disk or on a remote checkpointing server; in the event that a failure occurs, the application can be restarted from the last checkpoint. In combination with checkpointing, process migration can be used to deal with CPU unavailability or when a "better" host becomes available by moving the process to another machine. For example, the EXCL-PRED-CHKPT [7] is a scheduling policy that assumes a checkpoint frequency equals to 2.5 minutes (interval between two checkpoints) and the cost of checkpointing is 15 seconds (time to save/retrieve a checkpoint to/from a remote host). In this work, the authors note that the poor performance of EXCL-PRED-CHKPT is due to the fact that a task is not reassigned when it is assigned to a slow host or when the host becomes unavailable for task execution. Moreover, the authors consider that remote checkpointing or process migration is most likely infeasible in Internet environments, as the application can often consume hundreds of megabytes of memory and bandwidth over the Internet is often limited.

- Top = 19mm (0.75")
- Bottom = 43mm (1.69")
- Left = Right = 14.32mm (0.56")

Your paper must be in two column format with a space of 4.22mm (0.17") between columns.

D. *Economic models*

The scheduling policies previously described consider the resources in Computational Grid freely usable without any constraints. Unfortunately, if constraints are not put on how the resource on how resources are used, they may be misused. For instance, if all resources can be used without any limitation, situation in which all task are submitted to the best resources (e.g., the resource that has the best performance and availability) may arise, thus leaving idle other resources. Moreover, antisocial behaviors, where a user submits a replica of its application on all the resources, with the aim of speculatively exploit them to reduce the execution time of his application, may arise as well. In order to avoid these phenomena, a possible solution consists in associating a cost to each resource, and a budget to each user/application. When an application is run on a resource, a proper amount of (virtual) money is allocated to that application, and is subtracted from the total budget to account for the usage of the resource. The so-called economic approach to resource management has received great attention in the recent Grid literature [20], where both scheduling algorithms taking into account resource costs and application budgets [21], and automatic approaches for the computation of resource prices [22], have been proposed.

Some of the commonly used economic models that can be employed for managing resources environment include: the commodity market model, the posted price model, the bargaining model, the tendering/contract-net model, the auction model, the bid-based proportional resource sharing model, the community/coalition/bartering model and the monopoly and oligopoly. Details about each model are in [21].

IV. DISCUSSIONS

In this section, we will attempt to compare the scheduling algorithms previously described based on the taxonomy proposed.

A. *Information-based algorithm versus information-free algorithm*

Obtain information about the status of the resource is often difficult, especially in Grid environments. This is due to some characteristics that are intrinsic to Grids such as heterogeneity and volatility of the resources. Moreover, in order to achieve good performance, it is necessary to know the status of the resources in the next future, because just monitoring the resource (when it is possible) is not enough. The scientific literature has proposed tools able to obtain dynamic information such as host load and network bandwidth [23, 24, 25] with some good results [24, 25]. Unfortunately these are only initial encouraging results, thus the information-free scheduling algorithms are still popular in Grid environments.

B. *On-line scheduling versus batch mode scheduling*

The main difference, in terms of performance, between on-line scheduling and batch mode scheduling regards principally the arrival rate. When the arrival rate is low, machines may be ready to execute a task as soon as it arrives at the scheduler. Therefore, it may be beneficial to use the scheduler in the on-line mode so that a task does not need to wait the next scheduling event to begin its execution. In batch mode, the scheduler considers a bag of tasks for matching and scheduling at each scheduling event. This enables the scheduling algorithm to possibly make better decisions, because the scheduler have the resource requirement information for all tasks, and know about the actual execution times of a larger number of tasks (as more tasks might complete while waiting for the scheduling event). When the task arrival rate is high, there will be a sufficient number of tasks to keep the machines busy in between the scheduling events, and while an assignment is being computed.

C. *Fault-tolerant scheduling versus no fault-tolerant scheduling*

Although scheduling and fault tolerance have been traditionally considered independently from each other, there is a strong correlation between them. As a matter of fact, each time a fault-tolerance action must be performed, i.e. a replica must be created or a checkpointed job must be restarted, a scheduling decision must be taken in order to decide where these jobs must be run, and when their execution must start. A scheduling decision taken by considering only the needs of the faulty task may thus strongly adversely impact non-faulty jobs, and vice versa. Therefore, scheduling and fault tolerance should be jointly addressed in order to simultaneously achieve fault tolerance and satisfactory performance.

CONCLUSIONS

The intention of this paper has been to provide the related work in the area of resource management. This has been done through the presentation of taxonomy on the scheduling policies used in Grid computing. From our study, we can assert that usually the scheduling literature has considered efficiency and robustness as orthogonal aspects, that is their interactions are not taken into account when scheduling applications on Computational Grid. Unfortunately, as already mentioned before, in Desktop Grids faults may occur, and in this case the execution time of the applications gets much larger, as there is the need to recover from the fault. Consequently, there is the need of exploring scheduling strategies that attempts to maximize application performance in face of occurrence of faults.

REFERENCES

[1] [1] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, and C. Hwang. Characterizing and Classifying Desktop Grid. In CCGRID ’07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, pages 743–748, Washington, DC, USA, 2007. IEEE Computer Society

[2] [2] D. Kondo, A. Chien, and H. Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In Proc. of Super Computing Conference, 2004.

[3] [3] W. Cirne and et al. Grid computing for bag of tasks applications. In Proc. of 3rd IFIP Conf. on E-Commerce, E-Business and E-Government, 2003.

[4] [4] J. Smith and S. Srivastava. A System for Fault-Tolerant Execution of Data and Compute Intensive Programs over a Network of Workstations. In Proc. of EuroPar’96, volume 1123 of Lecture Notes in Computer Science, 1996.

[5] [5] D. da Silva, W. Cirne, and F. Brasileiro. Trading Cycles fro Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In Proc. of EuroPar 2003, volume 2790 of Lecture Notes in Computer Science, 2003.

[6] [6] R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauv_e. Fault in Grids: Why are they so bad and What can be done about it? In Proc. 4th Int. Workshop on Grid Computing (Grid 2003). IEEE-CS Press, Nov. 2003.

[7] [7] Derrick Kondo, Andrew A. Chien, Henri Casanova Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids. Proceedings of Super Computing Conference, 2004.

[8] [8] M. Maheswaran, S. Ali, H. Siegel, Debra Hensgen and R. Freund Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. Journal of Parallel and Distributed Computing, vol. 59, no. 2, pp. 107-131, 1999.

[9] [9] M. Mutka. Considering deadline constraints when allocating the shared capacity of private workstations. Int. Journal in Computer Simulation, vol. 4, no.1 pp. 4163, 1994

[10] [10] O. H. Ibarra and C. E. Kim Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. in the Journal of the ACM, vol.24, no.2, pp. 280{289, 1977.

[11] [11] A. Ghafoor and J. Yang A Distributed Heterogeneous Supercomputing Management System. IEEE Computer Society Press, vol.26, no.6, pp. 78{86, 1993.

[12] [12] M. Ka_l and I. Ahmad Optimal Task Assignment in Heterogeneous Distributed Computing Systems. IEEE Concurrency, vol.6, no.3, pp. 42{ 51, 1998.

[13] [13] M. Maheswaran and H.J. Siegel A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems. in Proceedings of the Seventh Heterogeneous Computing Workshop HCW '98, 1998.

[14] [14] H. Casanova, A. Legrand, and D. Zagorodnov et al. Heuristics for Scheduling Parameter Sweeping Application in Grid Environments. In Proc. of Heterogeneous Computing Workshop, IEEE CS Press, 2000.

[15] [15] M. Maheswaran and S. Ali and H. Siegel and D. Hensgen and R. Freund A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems. in the 8th Heterogeneous Computing Workshop, 1999.

[16] [16] E. Heymann and M. Senar and E. Luque and M. Livny Adaptive Scheduling for Master-Worker Applications on the Computational Grid In GRID, pp. 214-227, 2000.

[17] [17] D.P. da Silva, W. Cirne, and F.V. Brasileiro. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In Proc. of EuroPar 2003, volume 2790 of Lecture Notes in Computer Science, 2003.

[18] [18] J. H. Abawajy Fault-Tolerant Scheduling Policy for Grid Computing Systems. in Proceedings of the 18th International Parallel and Distribuited Processing Symposium (IPDPS'04), 2004.

[19] [19] J. Weissman and D. Womack. Fault Tolerant Scheduling in Distributed Networks. Technical Report TR CS-96-10, Department of Computer Science, University of Texas, San Antonio, Sept. 1996.

[20] [20] R. Buyya and M. Murshed GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. in the Journal of the Concurrency and Computation: Practice and Experience,vol. 14, no. 13-15, pp. 1175-1220, Wiley Press, USA, 2002.

[21] [21] R. Buyya and D. Abramson and J. Giddy and H. Stockinger Economic Models for Resource Management and Scheduling in Grid Computing. Journal of Concurrency and Computation: Practice and Experience (CCPE), Wiley Press, May 2002.

[22] [22] R. Wolski, J. S. Plank, J. Brevik and T. Bryan G-commerce: Market Formulations Controlling Resource Allocation on the Computational Grid. in the Book International Parallel and Distributed Processing Symposium (IPDPS), 2001.

[23] [23] P. Francis, S. Jamin, V. Paxson, L. Zhang and D.F. Gryniewicz and Y. Jin An Architecture for a Global Internet Host Distance Estimation Service. In the proocedings of IEEE INFOCOM, 1999.

[24] [24] B. Lowekamp, N. Miller, T. Gross, P. Steenkiste, J. Subhlok and D. Sutherland A resource query interface for network-aware applications. In the Journal of Cluster Computing, 1999.

[25] [25] R. Wolski Dynamically forecasting network performance using the network weather service. In the Journal of Cluster Computing, vol.1, no.1, pp.119-132, 1998.

[26] S. M. Metev and V. P. Veiko, *Laser Assisted Microtechnology*, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.

[27] J. Breckling, Ed., *The Analysis of Directional Time Series: Applications to Wind Speed and Direction*, ser. Lecture Notes in Statistics. Berlin, Germany: Springer, 1989, vol. 61.

[28] S. Zhang, C. Zhu, J. K. O. Sin, and P. K. T. Mok, “A novel ultrathin elevated channel low-temperature poly-Si TFT,” *IEEE Electron Device Lett.*, vol. 20, pp. 569–571, Nov. 1999.

[29] M. Wegmuller, J. P. von der Weid, P. Oberson, and N. Gisin, “High resolution fiber distributed measurements with coherent OFDR,” in *Proc. ECOC’00*, 2000, paper 11.3.4, p. 109.

[30] R. E. Sorace, V. S. Reinhardt, and S. A. Vaughn, “High-speed digital-to-RF converter,” U.S. Patent 5 668 842, Sept. 16, 1997.

[31] (2002) The IEEE website. [Online]. Available: http://www.ieee.org/

[32] M. Shell. (2002) IEEEtran homepage on CTAN. [Online]. Available: http://www.ctan.org/tex-archive/macros/latex/contrib/supported/IEEEtran/

[33] *FLEXChip Signal Processor (MC68175/D)*, Motorola, 1996.

[34] “PDCA12-70 data sheet,” Opto Speed SA, Mezzovico, Switzerland.

[35] A. Karnik, “Performance of TCP congestion control with rate feedback: TCP/ABR and rate adaptive TCP/IP,” M. Eng. thesis, Indian Institute of Science, Bangalore, India, Jan. 1999.

[36] J. Padhye, V. Firoiu, and D. Towsley, “A stochastic model of TCP Reno congestion avoidance and control,” Univ. of Massachusetts, Amherst, MA, CMPSCI Tech. Rep. 99-02, 1999.

[37] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, IEEE Std. 802.11, 1997.