

Semantic Scheduling of Active Measurements for meeting Network Monitoring Objectives

Prasad Calyam, Lakshmi Kumarasamy, Fusun Ozguner

Ohio Supercomputer Center/OARnet, The Ohio State University,
{pcalyam, lakshmi}@osc.edu; ozguner@ece.osu.edu

Abstract—Network control and management techniques (e.g., dynamic path switching, on-demand bandwidth provisioning) rely on active measurements of end-to-end network status. These measurements are needed to meet network monitoring objectives such as: (a) intra-domain/inter-domain paths status-checking, (b) network weather forecasting, (c) anomaly event detection and (d) fault-diagnosis. In this paper, we present a novel semantic scheduling algorithm based on deterministic and heuristic scheduling principles to handle measurement request loads for meeting network monitoring objectives that aid in resource adaptation decisions. The semantic priorities specified to our scheduling algorithm that are based on user-level and resource-level policies allow preferential treatment of measurement requests that supercede typical scheduling priorities based on periodicity and execution time. We evaluate our semantic scheduling algorithm using metrics such as cycle time and satisfaction ratio for increasing measurement request loads. Our semantic scheduling algorithm and evaluation study findings are vital to deploy and manage large-scale measurement infrastructures used for meeting monitoring objectives in support of next-generation applications and networks.

I. INTRODUCTION

Recent widespread deployment of openly-accessible measurement frameworks such as perfSONAR [1] has resulted in end-users competing for measurement resources. Hence, there is a dire need to develop semantic scheduling algorithms which allow precedence-based strategies for scheduling relatively more important measurement requests in the context of meeting network monitoring objectives.

Measurement requests from end-users feature diverse sampling time-interval patterns shown in Figure 1. Periodic sampling is required to produce accurate network weather forecasts [2] [3]. Whereas, random (e.g., poisson, exponential, gaussian) or stratified random sampling is used to detect network performance anomalies [4]. Further, there are frameworks such as [5] that use *adaptive* sampling for reducing the amount of processed data and to allow modifying the sampling frequency based on assumptions such as diurnal network patterns. The network status sampling is done using active measurement tools such as Ping, Traceroute, Iperf and Pathrate that inject a series of test packets into a network path and analyze their behavior to report metrics such as availability, delay, jitter, loss, route changes and bottleneck-bandwidth.

This material is based upon work supported by the Department of Energy under Award Number: DE-SC0001331. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

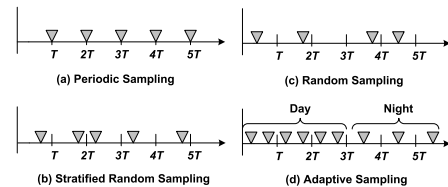


Fig. 1. Network Status Sampling Patterns

Anticipating above demands from end-users and also to serve their own routine monitoring purposes, Internet Service Providers have instrumented networks with measurement frameworks such as NLANR AMP [6], perfSONAR [1], Network Weather Service (NWS) [2] and ActiveMon [7] [3]. The measurement frameworks allow sampling of measurements at strategic points along inter-domain and/or intra-domain network paths. The measurement sampling is performed using *measurement schedulers* that invoke active measurement tools such that: (a) there are no “measurement conflicts” that lead to mis-reporting of network status due to CPU and channel resource contention from concurrently executing tools, and (b) active measurement probe traffic is regulated based on pre-specified “measurement level agreements” (MLAs) [3] (e.g., upto 5% of network bandwidth can be probe traffic).

Given that measurement resources (i.e., tool servers, bandwidth) are limited, feasible schedules may not be possible under high measurement request loads. Consequently, measurement requests that could not be scheduled might adversely affect monitoring accuracy. Our work is motivated by the fact that none of the measurement scheduling algorithms used in existing measurement frameworks have the ability to handle semantic priorities that allow preferential treatment of measurement requests when end-users compete for using measurement resources. In this paper, we present a novel semantic scheduling algorithm that is conflict-free and MLA-compliant, and handles measurement requests with semantic priorities for meeting network monitoring objectives that aid in resource adaptation decisions. The algorithm is based on deterministic and heuristic scheduling principles and uses semantic priorities driven by user-level and resource-level policies.

The remainder paper organization is as follows: Section 2 presents a background on measurement requests and their handling by scheduling algorithms. Section 3 provides details of our novel semantic scheduling algorithm and related schedule generation issues. Section 4 presents performance evaluations of our algorithm, and Section 5 concludes the paper.

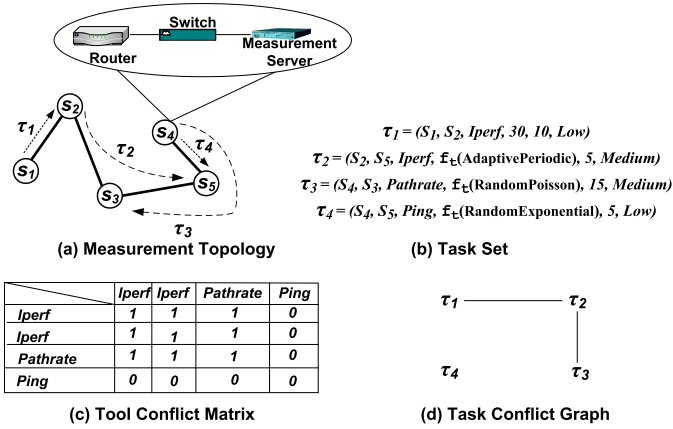


Fig. 2. Example Task Set and Conflict Graph

II. BACKGROUND AND TERMINOLOGY

A. Measurement Tasks

The network paths to be measured are specified by a *measurement topology*, which can be formally represented by a graph $G = (N, E)$, where N is the set of measurement servers and E is the set of edges between a pair of measurement servers. On top of the measurement topology, a set of on-going measurement requests are specified. Using real-time systems terminology, an on-going measurement request can be treated as an “offline task” τ_i . The τ_i specification contains the source server src_i , destination server dst_i , active measurement tool $tool_i$, inter-sampling time between consecutive measurement instances based on the desired sampling pattern shown in Figure 1, execution time e_i of a single measurement instance, and finally the semantic priority. Figure 2 shows the measurement topology of an example task set. A task τ_i with a fixed inter-sampling time or period (p_i) can be expressed as follows:

$$\tau_i = (src_i, dst_i, tool_i, p_i, e_i, sp_i).$$

For non-periodic tasks, the inter-sampling time p_i follows a function $f(s)$ corresponding to their sampling pattern. We denote the j -th instance (or job) of τ_i as τ_{ij} . Further, the time when the j -th job τ_{ij} is released to be scheduled is called the *release time* and simply given by $(j - 1)p_i$. The tool conflict matrix of a task set will have an entry of 1 if any tool is empirically found to be CPU or channel intensive, and if it causes measurement conflict with another tool that has similar resource consumption characteristics. The task conflict graph of a task set contains edges between conflicting tasks i.e., they have a common measurement server or measurement path. We define *hyperperiod* for task set as the least common multiple of all the task periods in the set. Alternately, the schedule for all the input tasks is generated over a hyperperiod duration.

B. Measurement Scheduling Algorithms

Heuristic scheduling algorithms similar to Heuristic Bin Packing (HBP) [7] are efficient in scheduling routine network monitoring tasks, where the *execution time* of a task determines its relative scheduling priority. The HBP algorithm packs the non-conflicting and MLA-compliant measurement

jobs in schedule bins using a “first-fit decreasing” [9] principle, where jobs are ordered with decreasing execution times, and jobs with higher execution times are scheduled earlier than jobs with relatively lower execution times.

Deterministic scheduling algorithms similar to EDF-CE [3] [8] are efficient in scheduling network weather forecasting related measurement tasks, where task *periodicity* (equivalently the task’s deadline) determines its relative scheduling priority. The EDF-CE scheduling algorithm orders and schedules measurement requests based on the earliest deadline first (EDF) scheduling principle, while allowing concurrent execution if jobs do not conflict nor violate MLAs.

III. SCHEDULING SEMANTIC PRIORITY TASKS

A. Semantic Scheduling Algorithm

The semantic priority sp_i in a measurement task τ_i is specified when there is an urgency to schedule the task by superseding typical scheduling priorities i.e., period and execution time. The sp_i can be based on “user-level” policies (e.g., a certain user role has higher privileges to submit measurement requests compared to other user roles) or “resource-level” policies (e.g., measurement requests involving intra-domain resources are more important than those involving inter-domain resources). To determine sp_i , we could ideally define an ontology to describe measurement tasks and apply inference techniques. However, we choose a simple approach in this paper and abstract the semantic priorities into three grades: *high*, *medium*, and *low*.

Herein, we describe the Semantic Priority Scheduling (SPS-GPE) algorithm that schedules measurement tasks with semantic priority grade inputs. The ‘GPE’ term indicates that the scheduler first considers semantic priority grade followed by periodicity and execution time, respectively while generating measurement schedules. For a given task set with GPE inputs, the SPS-GPE algorithm determines the relative priority of a measurement task on a scale of 0 to 2 considering the characteristics of the overall task set using a “penalty function”. We now explain the sequence of steps involved in the SPS-PE algorithm with penalty function calculations as illustrated in Figure 3, and follow it up with an explanation of how the SPS-PE algorithm is transformed into the SPS-GPE algorithm.

The SPS-PE algorithm combines the features of both the deterministic and heuristic scheduling algorithms. The penalty function calculation in the SPS-PE algorithm is as follows: Initially, all tasks are assigned a priority of zero. Next, we obtain the maximum execution time among the tasks in the given task set. Subsequently, for each task we calculate the relative execution time of the task as, execution time of the task divided by the maximum execution time, and increment the priority using this calculated value. We then check if a task has periodicity (i.e., periodic, stratified random or adaptive with periodic/stratified random segments) in its specification. If the task has periodicity, we increment the already assigned priority by 1.

Once the priority of each task is calculated by the penalty function, the tasks are arranged in decreasing order of their priority. We move the periodic jobs (pure periodic, stratified random or adaptive with periodic segments) into the EDF

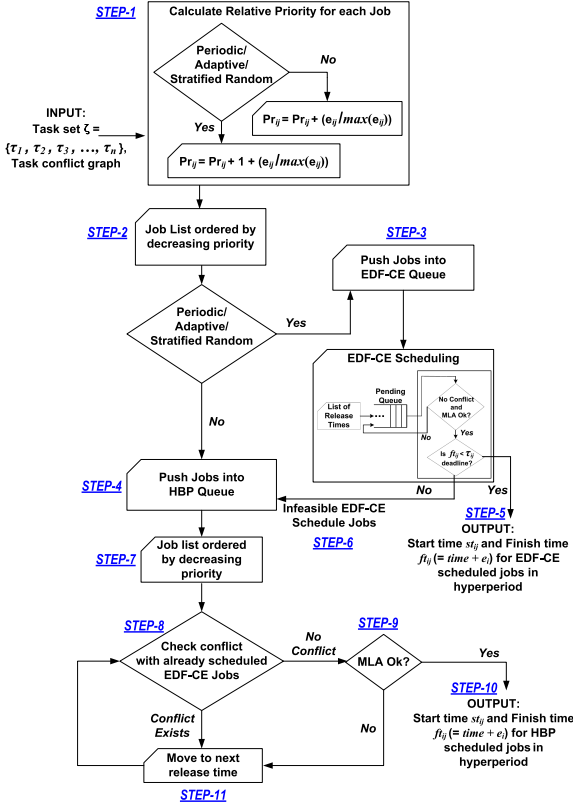


Fig. 3. SPS-PE Schedule Generation Flowchart

queue, and the random sampling pattern jobs into the HBP queue. We then schedule the periodic jobs in EDF queue using the EDF-CE algorithm and move the jobs that result in an infeasible schedule into the HBP queue. The position of these jobs in the queue depends on their priority. For each job in the HBP queue, we check if it conflicts with the already scheduled EDF-CE jobs. If they do not conflict and if MLA constraint is not violated, we schedule the job with the same release time. If any of the above conditions are not met, we move the job to the next release time and continue to do so until the job is scheduled.

To transform the SPS-PE into the SPS-GPE algorithm, the initial step of ordering the tasks is based on the user-level and resource-level policies (high, medium and low grades) associated with the tasks. In case of a tie between same grade tasks for schedule generation (e.g., 2 tasks having high grades), we consider their corresponding scheduling priorities calculated using the penalty function to re-order the tasks. The jobs of the re-ordered tasks are then pushed into the HBP and EDF-CE queues and are scheduled similar to the sequence of steps of the SPS-PE algorithm shown in Figure 3. Unlike SPS-PE algorithm which moves the jobs in HBP queue to the next release time, the jobs with low priority (either EDF-CE or HBP scheduled jobs) are moved to the next release time in case of SPS-GPE algorithm.

B. Semantic Schedule Generation Issues

Figure 4 shows the schedule output for the tasks in Figure 2 using the SPS-PE algorithm. Note that upward arrows in

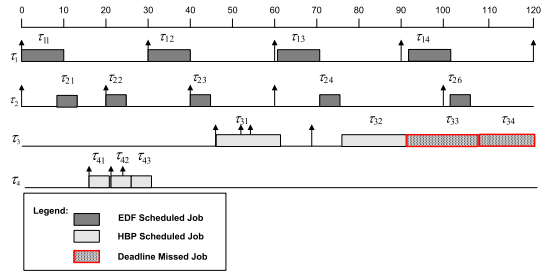


Fig. 4. Semantic PE Schedule

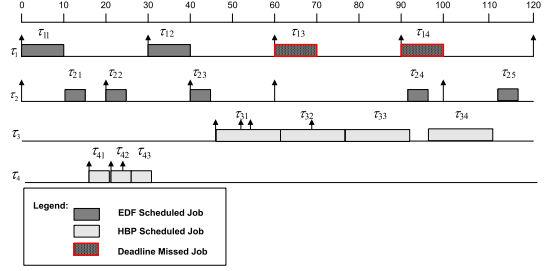


Fig. 5. Semantic GPE Schedule

the figure indicate the release times of measurement jobs. By calculating priority as described in the earlier subsection III-A, we assign the priorities of tasks as follows: $\tau_1 = (10/15)+1=1.66$; $\tau_2 = (5/15)+1=1.33$; $\tau_3=(15/15)=1$; $\tau_4=(5/15)=0.33$. Jobs of tasks τ_1 and τ_2 are scheduled before their deadline because of their higher scheduling priorities compared to τ_3 and τ_4 . Jobs of task τ_4 are scheduled as soon as they arrive because they do not conflict with either τ_1 , τ_2 or τ_3 tasks. Due to their lower scheduling priority, jobs of task τ_3 i.e., τ_{33} and τ_{34} are not assigned a slot within their deadlines. However, SPS-PE schedules them in a later time slot towards the tail-end of the final schedule without totally discarding them.

Figure 5 shows the schedule output for the tasks in Figure 2 using the SPS-GPE algorithm. Both tasks τ_2 and τ_3 have ‘medium’ semantic grade priority, and hence in this priority tie case, we check their scheduling priorities. Since τ_2 has higher scheduling priority compared to τ_3 , task τ_2 gets highest preference, followed by tasks τ_3 , τ_1 and τ_4 , respectively. Jobs of task τ_4 in this case also get scheduled as soon as they arrive because they do not conflict with either τ_1 , τ_2 or τ_3 tasks. Since tasks τ_2 and τ_3 have higher priority than task τ_1 , the SPS-GPE schedules jobs of τ_2 and τ_3 before their deadlines, and consequently causes deadline misses of jobs τ_{13} and τ_{14} .

IV. PERFORMANCE EVALUATION

A. Simulation Setup

Our synthetic task set is similar to the example task set shown in Figure 2. The period p_i and execution time e_i of each task τ_i is randomly selected from [1000 sec, 10000 sec] and [100 sec, 999 sec], respectively. The period p_i and execution time e_i ranges are based on values of commonly used tools in existing measurement frameworks such as perfSONAR and Activemon. To evaluate our algorithm scalability, we consider

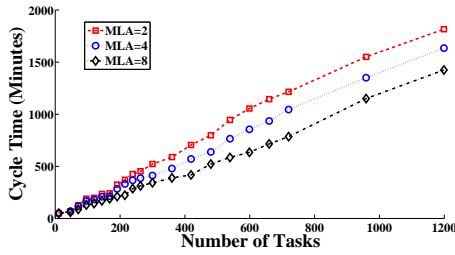


Fig. 6. Cycle time trends for increasing MLAs and tasks

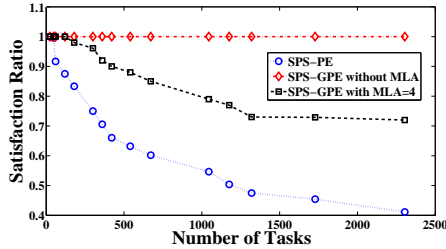


Fig. 7. Comparison of SPS-PE and SPS-GPE

2500 servers in our simulations which is a realistic number in large-scale multi-domain measurement infrastructures. The type of sampling pattern is chosen randomly from the set of sampling patterns comprising of periodic, stratified random, random (i.e., poisson, exponential, gaussian) and adaptive.

For each task set and corresponding task conflict graph, we average the results obtained by running the simulation over 500 iterations to obtain the data points shown in evaluation graphs. We use *Cycle Time* and *Satisfaction Ratio* as performance metrics in our evaluations. *Cycle time* is defined as the duration over which a complete round of all scheduled jobs are executed. By minimizing the cycle time during schedule generation, measurements can be initiated as frequently as possible for obtaining up-to-date network-wide status snapshots. *Satisfaction Ratio* is defined as shown in Equation (1). Satisfied users are those whose monitoring objectives are successfully met by a measurement scheduling algorithm, assuming: (a) each task is input by a unique user, and (b) monitoring objective is met if all the jobs of a task are scheduled prior to their deadlines.

$$\text{Satisfaction Ratio} = \frac{\text{No. of satisfied users}}{\text{Total No. of users}} \quad (1)$$

B. Scheduling Performance Characterization

1) *Effect of MLAs*: Figure 6 shows the effect of MLAs on cycle time trends for increasing number of input tasks. We can see that - as the MLA constraint value (given by maximum number of concurrent execution jobs permitted network-wide) increases, the cycle time decreases. We observe that MLA constraint is a bottleneck in generating a highly efficient semantic schedule. We can also see that - as the number of tasks increases, the cycle time also increases correspondingly.

2) *Effect of Semantic Grades*: Herein, we show how semantic priority grade in the SPS-GPE algorithm over-rides the

scheduling priorities of period and execution time. In the SPS-PE algorithm, a priority of 2 denotes highest priority; periodic tasks get a default priority of 1, and a relative priority of 0 to 1 based on their execution time. This causes periodic tasks, especially the ones with high execution times to get higher priority and thus have the best chance to be scheduled before their deadlines. However, in the SPS-GPE algorithm, tasks with ‘high’ semantic priority grade get highest priority and get scheduled first, followed by ‘medium’ and ‘low’ grade priority tasks (irrespective of task’s scheduling priority based on period and execution time). Thus, if all the high priority are scheduled before their deadline, we consider the satisfaction ratio to be 1. For the SPS-PE algorithm, satisfaction ratio depends on the job deadline misses.

As shown in Figure 7, if there are no MLA constraints, satisfaction ratio using the SPS-GPE algorithm is always equal to 1 even with large number of task inputs. In case of SPS-PE algorithm, we can see that, satisfaction ratio drops drastically below 1 with increasing number of tasks. We can also infer that if there are excessive MLA constraints or inter-task conflicts represented by MLA=4 condition, satisfaction ratio of SPS-GPE drops below 1. Nevertheless, satisfaction ratio of the SPS-GPE algorithm is always higher than the corresponding value in the case of the SPS-PE algorithm.

V. CONCLUSION

In this paper, we proposed a novel semantic priority scheduling (SPS-GPE) algorithm based on deterministic and heuristic scheduling principles to allow preferential treatment of relatively more important measurement requests when end-users compete for measurement resources. Our evaluations showed that the SPS-GPE algorithm produces efficient measurement schedules (low cycle times) and successfully schedules high priority measurement requests (high satisfaction ratios) for increasing measurement request loads.

REFERENCES

- [1] A. Hanemann, J. Boote, E. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, M. Swamy, S. Trocha, J. Zurawski, “PerfSONAR: A Service Oriented Architecture for Multi-Domain Network Monitoring”, *Proc. of Service Oriented Computing, Springer Verlag, LNCS 3826*, pp. 241-254, 2005. (<http://www.perfsonar.net>)
- [2] B. Gaidioz, R. Wolski, B. Tourancheau, “Synchronizing Network Probes to avoid Measurement Intrusiveness with the Network Weather Service”, *Proc. of IEEE HPDC*, 2000.
- [3] P. Calyam, C.-G. Lee, E. Ekici, M. Haffner, N. Howes, “Orchestrating Network-wide Active Measurements for Supporting Distributed Computing Applications”, *IEEE Transactions on Computers*, 2006.
- [4] P. Calyam, J. Pu, W. Mandrawa, A. Krishnamurthy, “OnTimeDetect: Dynamic Network Anomaly Notification in perfSONAR Deployments”, *Proc. of IEEE/ACM MASCOTS* (to appear), 2010.
- [5] W. Ma, J. Yan, C. Huang, “Adaptive Sampling Methods for Network Performance Measurement under Voice Traffic”, *Proc. of IEEE ICC*, 2004.
- [6] A. McGregor, H-W. Braoun, “Automated Event Detection for Active Measurement Systems”, *Proc. of Passive and Active Measurement Workshop*, 2001.
- [7] P. Calyam, C.-G. Lee, P. K. Arava, D. Krymskiy, D. Lee, “OnTimeMeasure: A Scalable Framework for Scheduling Active Measurements”, *Proc. of IEEE E2EMON*, 2005.
- [8] Z. Qin, R. Rojas-Cessa, N. Ansari, “Task-execution Scheduling Schemes for Network Measurement and Monitoring”, *Elsevier Computer Communications*, Volume 33, Issue 2, Pages 124-135, 2010.
- [9] G. Ausiello, P. Crescenzi, V. Kann, et. al., “Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties”, *Springer Publication ISBN:3540654313*, 1998.