Deadline-Aware Co-Scheduling Using Anycast Advance Reservations in Wavelength Routed Lambda Grids

Hitesh Kulkarni, Arush Gadkar, and Vinod M. Vokkarane
Department of Computer and Information Science
University of Massachusetts Dartmouth
Email: vvokkarane@ieee.org

***Abstract*—As grid applications evolve, the need for user controlled network infrastructure is apparent in order to support emerging dynamic and interactive services. Due to the inherent bandwidth offered by optical wavelength division multiplexed (WDM) networks, they prove to be a potential candidate to support the bandwidth intensive grid applications. In a grid environment, the available resources can be classified into two broad categories: *grid resources* which consist of computing and storage components that reside on each node of the network and *network resources* which provide bandwidth for the execution of a grid application. A typical grid application (job) is usually divided into a number of smaller tasks which need to be scheduled, on possibly different nodes of the network in order to ensure job completion. There usually exists some dependency between these tasks and a strict time-deadline within which the job needs to be completed. Rather than using an independent scheduling approach (at the grid and network levels), we address the *co-scheduling* problem in lambda-grids for advance reservation requests and aim at minimizing the job blocking probability. We use the anycasting communication paradigm referred to as *co-anycasting*, to allocate grid *and* network resources to all tasks of a job. We propose three heuristics: *first free (FF)*, *shortest hop (SH)*, and *least used (LU)* to solve the co-scheduling problem. Moreover, we compare the proposed *co-anycasting* approach to a *grid-anycasting* approach, wherein the anycasting flexibility is offered only at the grid level, and show through extensive simulations the performance benefit of using *co-anycasting* to support grid applications in a time-deadline environment.**

***Index Terms*—Lambda-Grid, Co-Scheduling, Grid Computing, Network-Bandwidth, Co-Anycasting, Advance Reservations.**

I. Introduction

Recently, a growing number of scientific applications require large amounts of data (usually in the scale of petabytes) generated by experiments to be accessible and analyzed by a large number of geographically dispersed users. Such large-scale applications can easily overwhelm the computing systems capacity of an organization. Hence the need for cooperation of computing devices from different organizations to run these applications is critical. The grid can help to solve this problem. A grid, simply defined, is the integration of distributed facilities that offer computing and storage resources to the clients as a service.

In order to efficiently realize such an environment, one must also take into account the support from underlying networks, which are required to provide high capacity and communication capabilities that are service oriented. Due to the inherent bandwidth offered by optical wavelength division multiplexed (WDM) networks, they prove to be a potential candidate for such a backbone network. This integration of grid resources interconnected by WDM fiber links is termed as *Lambda-Grid*. A key difference between a *Lambda-Grid* request and the traditional network connection request is location transparency. In a network connection request, the location (destination node ID) needs to be specified. This request is provisioned by finding an end-to-end route over the physical topology on a particular wavelength[1]. This is known as the *routing and wavelength assignment (RWA)* problem and is known to be NP-Complete [1]. The combination of the physical route and wavelength is called a *lightpath*. On the other hand, a grid request supports location transparency, wherein the request specifies only the requirement of resources, but not the exact location or ID of the destination node. Hence, a grid request is provisioned in two steps: 1) finding a node with sufficient available resources that can be allocated to the request and 2) establishing a lightpath (from the source node, i.e., the node where the request originates) to the selected destination node. This problem of jointly finding a node (with sufficient resources) and establishing a lightpath to it is termed as *co-scheduling*.

A grid request, referred to as a *job* can be divided into a number of modules, referred to as *tasks* that can be dependent on each other. Dependent tasks of a job are modeled using a directed acyclic graph (DAG) [2], wherein the nodes of the graph denote the task and the arcs denote the dependency between the tasks. Hence a particular job can be represented by a uni-directional graph. A user's grid request can be broadly classified into two categories: *immediate-reservation (IR)* and *advance-reservation (AR)* requests. IR requests are provisioned (if possible) immediately upon their arrival, i.e., the scheduler checks if sufficient resources (grid and network) are available to provision the request, else it is deemed blocked. On the other hand, AR requests provide a time-deadline within which the request can be provisioned. This flexibility provides the ability to the scheduler to guarantee the availability of resources at a particular time in the future. This increases the predictability of the system and results in improved performance. Note that in the context of AR jobs (request), each task of the job needs to be scheduled within the specified time-deadline, taking into account the task dependencies, i.e., for example, if task B is dependent on task A, we need to ensure that prior to scheduling task B, we have successfully completed scheduling task A. We refer to this as the *deadline-aware co-scheduling in lambda-grids (DACL)* problem.

[1]Using the same wavelength on all the physical links a connection traverses, helps avoid the need to incorporate wavelength converters which are expensive. This property is commonly referred to as the *wavelength continuity constraint* (WCC).

978-1-4673-0009-4/13/$26.00 ©2013 IEEE

In this paper, we use the *anycasting* communication paradigm to solve the *DACL* problem. Anycasting is defined as the communication paradigm in which the user, has the ability to choose a single destination from a group of candidate destinations. Note that in the context of Lambda-Grids, this functionality is offered, both at the grid and network levels. We refer to this as *co-anycasting*. We propose three heuristics and aim at minimizing the job blocking probability. We show how time-deadline affects the performance of the grid and how the resources available on the node are utilized using these anycasting algorithms. Moreover, we also compare the *co-anycasting* approach to solve the DACL problem to a *grid-anycast* approach, wherein the anycasting functionality is offered *only* at the grid level. Our study establishes the effectiveness of using the anycasting communication paradigm to schedule resources in a time-deadline environment. The remainder of this paper is organized as follows: we discuss the related work in Section II and formally define the problem in Section III. We present the anycasting heuristics in Section IV. In Section V, we present the results of our performance evaluations and finally conclude the paper in Section VI.

II. Related Work and Motivation

There is a lot of work in the literature on scheduling methods in both optical networks [3] and grid computing [4]. However these methods are too restrictive to be directly used to achieve optimal grid-network scheduling. In [5], the authors address the problem of co-scheduling in grid networks and propose various job scheduling heuristics, such as greedy, earliest start time first (ESTF), largest job first (LJF), and random selection. They show that adaptive routing schemes perform better than using a fixed routing method. A Tabu search algorithm to perform joint scheduling is proposed by the same authors in [6].

In [7] and [8], the authors propose various anycast algorithms to improve the job acceptance rate, in the context of grid over OBS networks. [7] considers jobs which have tasks dependent sequentially on each other and proposes various cost assignment policies to pick up a combination of nodes, providing the required services and routes based on a *n-partite graph*. [9] and [10] have considered advanced reservations on grid-networks. In [9], the authors have considered advanced reservation of resources for both homogeneous and heterogeneous environments. The authors provided scheduling algorithms using best fit strategy to utilize the maximum amount of resources. [10] addresses the resource provisioning in WDM networks by providing ILPs and heuristics for maximizing the number of connections accepted and minimizing the number of wavelength links. Note that these works address the problem of advanced reservations on grid networks only, and do not solve the joint problem (co-scheduling) in grid networks.

Various frameworks are built for grid and network co-scheduling. *PHOSPHORUS* [11] and *EnLIGHTened* [12] are projects focusing on providing on-demand and in-advance advanced reservation for grid resources. G-Lambda [13] is another project with a goal of providing a standardized web interface between grid resource scheduler and network resource management systems. Practical grid applications, such as the large-scale scientific experiments (Large Hadron Collider) and high energy particle physics applications (Fusion Energy Science) have strict time-deadlines within which a particular job needs to be completed. With this in mind, we extend the work conducted in [5] and propose efficient heuristics to minimize the job blocking probability (for AR requests) in a time-deadline driven environment. Moreover, in [5] the authors imposed a limitation, that every computing node can execute only one task at a time and only a sequential dependence between the tasks was assumed. In this work, we assume a random dependence between the tasks and allow multiple tasks to be executed on a particular node, provided it has sufficient resources.

III. Problem Definition and Methodology

In this section we formally define the DACL problem and discuss the assumptions used in this paper. Given a network $G = (V, E, R_v, W)$, (V is the set of nodes, E is the set of edges, W is the number of wavelengths per fiber, and R_v is the number of resources available on node $v \in V$), we consider a time-slotted network with fixed-size time slots. A job is in form of a DAG which can be modeled as $J = (T, A)$, where T is the set of tasks and A specifies the set of arcs which connect the tasks in the manner of their dependencies. Each task can be further defined as $T_j = (T_{aj}, D_{uj}, S_j, R_j, D_{lj})$, where T_{aj} is the arrival time of the task j, D_{uj} is the duration, S_j is the source node[2], R_j is the number of resources required by the task, and D_{lj} is the deadline for task scheduling. A network scheduler maintains the state information which is updated for every new task. The network scheduler's state information consists of which time slots are in use on all the wavelengths and on all links across the network. It can be thought of as a table $U[E, W]$, where E is the set of links and W is the set of wavelengths on that link. A similar information of which resources are occupied across all the nodes is maintained by the grid scheduler. With this in mind the DACL problem can be formally stated as follows:

Definition (DACL): Given a network $G = (V, E, R_v, W)$, and a dynamic arrival process for the tasks (that comprise a job), one must assign sufficient resources for each task and schedule a lightpath from the client node to the resource node in such a manner that the number of jobs scheduled is maximized while satisfying the wavelength continuity constraint and adhering to the time-deadline constraint of each task.

Methodology: The tasks of a job follow a dependency graph. The grid scheduler picks up a job from the job list and selects a task which has no parent in order to schedule it. The grid scheduler then checks if there are any nodes available with the number of resources, greater than or equal to the resources required by the task to be executed. All the node's which are available with the number of resources required, are added to the set of potential destination nodes given by G_l. This destination list, G_l is then passed on to the network

[2]For sake of simplicity, we use the following definition through the remainder of this paper: The source node is called as the *client* node i.e., the node where the request originates and we denote the node which services this client node with the required number of resources as the *resource* node.

scheduler in order to find a ligthpath from the client node to a possible resource node. The network scheduler then evaluates (depending on it's current state), the possible RWA's that can be successfully achieved to the nodes in G_l, and creates a list of those nodes, D_{lj}. Then depending upon the flavor of the anycast algorithm (described in the next section), a destination, D_j, is picked from D_{lj}. Note that this destination D_j becomes the client node for every task (child) dependent on this current task.

We assume following constraints for scheduling tasks on grid and network level.

- Before scheduling a task, all of its parent tasks should be completed.
- For any task T_i, its deadline, D_{li} is greater than the maximum deadline from the set of deadlines of its parent task(s).
- The task cannot be scheduled to be executed on its source node, i.e., $S_i \neq D_i$.
- The duration of time required by the task, D_{ui}, is same for the lightpath and resources on destination (resource) node D_i.
- If a task T_i belonging to job J is blocked, all the consecutive tasks of J are blocked, thereby blocking the job.
- There are no wavelength converters in the network, so any given task must use the same wavelength on all the links on its lightpath, i.e., we adhere to the WCC.

Every task after being scheduled can be represented as $T_j = (T_{aj}, D_{uj}, S_j, D_j, R_j, D_{lj}, P_j, W_j)$, where D_j is the selected destination, P_j is the lightpath scheduled from S_j to D_j, and W_j is the wavelength chosen for transmission. Note that we do allow multiple tasks to be executed concurrently on a given node and aim at minimizing the job blocking probability.

IV. Anycast Co-Scheduling Heuristics

In this section, we present the anycast heuristics for grid and network co-scheduling. Our algorithms employ a fixed routing scheme. We first present three different anycast heuristics for resource node selection: *first free (FF)*, *shortest hop (SH)*, and *least used (LU)* and then describe the co-scheduling algorithm.

A. First Free (FF)

We denote all the nodes of the network by some node ID (index number) and include them in a set N. To schedule a particular task, we select the nodes from N which have sufficient resources (as that required by the task) and add them to a set G_l ($G_l \subset N$). G_l is then sorted according to the index number of the nodes in an increasing order and passed to the network scheduler in order to perform an RWA, i.e., to check if a lightpath can be established from the client node to the current node under consideration in G_l. If the network scheduler cannot successfully schedule a lightpath to any node in G_l, the task is deemed blocked. Note that the *FF* heuristic does not incorporate any network/grid parameter to choose the resource node, in contrast to the two heuristics proposed below.

B. Shortest Hop (SH)

The *SH* heuristic finds all the possible nodes that satisfy the task's grid resource requirements and adds them to the set G_l, which forms the anycast destination set. G_l is then sorted based on the shortest hop count (from the client node) in an increasing order. The heuristic then scans all the paths and wavelength combinations to find all the available lightpaths with duration D_{ui} (of task T_i) for each destination in the anycast destination set. The heuristic selects a destination with the shortest hop count and tries to schedule a lightpath to it. If the requested number of time slots are not available on any path connecting the client node, S_i, to this current node under consideration in G_l, the network scheduler moves to the next node and tries to schedule a lightpath to it, and so on until the all the nodes from G_l are exhausted. If a lightpath cannot be scheduled the task is deemed blocked, thus blocking the job.

C. Least Used (LU)

The *LU* heuristic finds all the possible nodes that satisfy the task's grid resource requirement and adds them to the set G_l, in an increasing order of the tasks which are currently being executed on them. This heuristic thus ensures that all the nodes participating in the grid are used, which in turn helps reduce the blocking probability from the network layer perspective. The network scheduler then scans all the nodes from G_l and tries to schedule a lightpath to any *one* node. If an RWA cannot be achieved to any node in G_l, the task, and in turn the job, is deemed blocked.

D. Co-scheduling Grid and Network Resources

Algorithm 1 depicts the co-scheduling algorithm implemented in this paper. For a particular job, J, (under consideration to be scheduled) the grid-scheduler selects a task, T, in such a way that its dependency is satisfied. Depending upon the current availability, we populate a list, G_l, with nodes that can satisfy the resource requirement of T (as shown in Lines $1-4$). To populate G_l, we also consider nodes that do not have the required number of grid resources free at the present time (task's arrival time), but would have free resources at some time in the future, before the deadline of the task is reached (as shown in Lines $6-8$). In such cases an advance reservation is performed on the grid and network resources. Once all the nodes have been added to the set G_l, we sort them (Line 9) according to the policies discussed earlier (first free, shortest hop, and least used) and pass them to the network scheduler to perform the RWA operation. Note that the request is blocked at the grid level in case G_l is empty. The RWA function determines how many consecutive time slots are available starting at the arrival time (T_a) on a specific lightpath, or it does an advanced reservation in case the grid destination is to be scheduled for $(T_a + SA)$, where SA is the number of slots from the arrival time slot after which allocation should occur. The *co-schedule()* function (Line 15) schedules and reserves the resources on network and grid level and holds them until their completion times. A request is deemed blocked at the network level only when a successful RWA cannot be

performed on *any* node in G_l. Finally, when all the tasks of a particular job are completed, the job is marked completed.

Algorithm 1: Co-Scheduling Algorithm

Input: $T_j = (T_a, D_u, S, R, D_l)$, $G = (V, E, R_v, W)$, $U[E, W]$
Output: *co-schedule*(T) *where* $T = (T_a, D_u, S, D, R, D_l, P, W)$

```
for v = 1 to V and v ≠ S do
    R_v f = Check_Free_Now(v)
    if R_v f ≥ R then
        G_l.add(v)
    else
        R_v f = Check_Free_Before_Deadline(v, D)
        if R_v f ≥ R then
            G_l.add(v)
Sort Anycast Destinations G_l based on the anycast algorithm(SH, LUorFF)
then
for d = 1 to G_l do
    for w = 1 to W do
        if RWA(S, G_l d, w, P) then
            T = (T_a, D_u, S, G_l d, R, D_l, P, w)
            co-schedule(T)
```

V. Performance Evaluation

We evaluate our anycast heuristics on the 14-node National Science Foundation network (NSFnet), shown in Fig. 1. Jobs arrive according to a Poisson process with average arrival rate λ and have exponentially distributed holding times with an average service rate of μ. Note that the time-deadline for the jobs are also generated exponentially with service rate greater than μ. The horizon is large enough so that none of the requests are blocked due to their durations. The number of tasks per job is uniformly distributed across the range $[1, 5]$. For each job, the dependencies between the tasks are randomly generated. Note that the tasks with no parents have randomly generated source nodes. The time slot size greatly depends upon the type of traffic that a network operator expects. It could be fine tuned by the operator to range from seconds to hours. We simulate 10^5 jobs per run and plot an average of 30 runs in all the following figures.

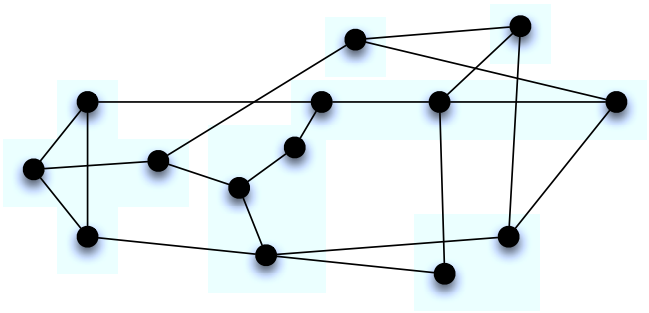
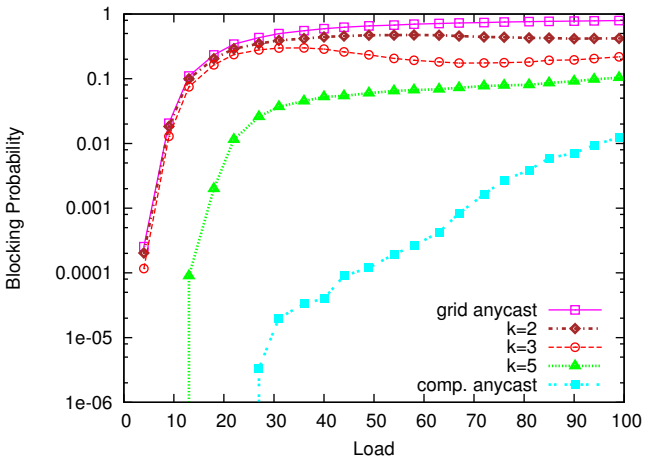
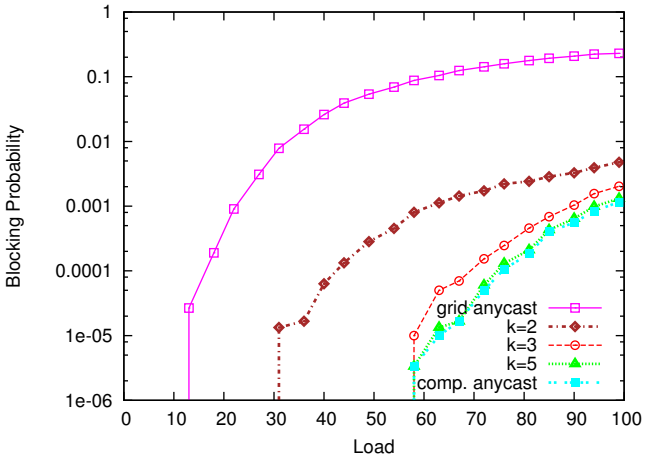


Fig. 1. Simulation topology: NSFnet.

We first evaluate the benefits of providing the anycasting functionality, both at the grid and network level (referred to as *co-anycast*) as opposed to providing it only at the grid level (referred to as *grid-anycast*). In the anycast communication paradigm, the client node communicates with *any one* resource node from the set of candidate nodes (G_l). We further use the parameter $k = \{2, 3, 5\}$ to represent the number of nodes



(a)



(b)

Fig. 2. Co-Anycasting vs Grid-Anycasting (a) FF (b) SH.

(from the set of $N - 1$ nodes) that can be potentially added (depending upon resource availability) to the set G_l. We also evaluate our heuristics for the case of *complete-anycast*, wherein we do not impose a limit on the number of nodes that can be added to the set G_l, i.e., we can potentially add, depending on resource availability, up to $k = N - 1$ nodes. With this in mind, the *grid-anycast*, communication paradigm may be defined as one wherein we impose the restriction of $k = 1$. In doing so, we provide flexibility only at the grid level[3]. No flexibility is offered at the network level, as an RWA, *only* to the selected node has to be successfully performed to provision the request. In Fig. 2, we compare the benefits of co-anycasting to the case of *grid-anycast*, for the *FF* and *SH* heuristics respectively. It can be observed that all flavors of *co-anycast* (i.e., $k = 2, 3, 5$ and *complete-anycast*) outperform the *grid-anycast*, by an order of magnitude for the the *FF* and by almost two-orders of magnitude for the the *SH* heuristic across different loads[4]. As expected, with an increase

[3]Like the unicast (one-to-one) communication paradigm, the *grid-anycast* also consists of a single node with which the client node communicates. However, we use the policies (*FF, SH, and LU*) to chose which node the client communicates with, unlike deciding it a-priori as in unicast.

[4]The network load in Erlangs is calculated as the ratio of the average arrival rate to the average service rate (λ/μ).

in k (number of nodes in the set G_l, an improvement in the blocking performance is observed as more flexibility is granted to the co-scheduling algorithm to provision the requests. These results justify the performance benefits achieved by incorporating the anycasting functionality at both the grid and network level i.e., *co-anycasting* as opposed to *grid-anycast*.

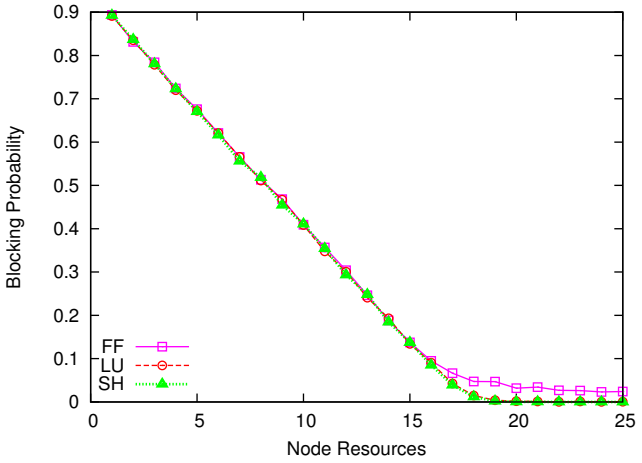


Fig. 3. Blocking performance for different number of node resources.

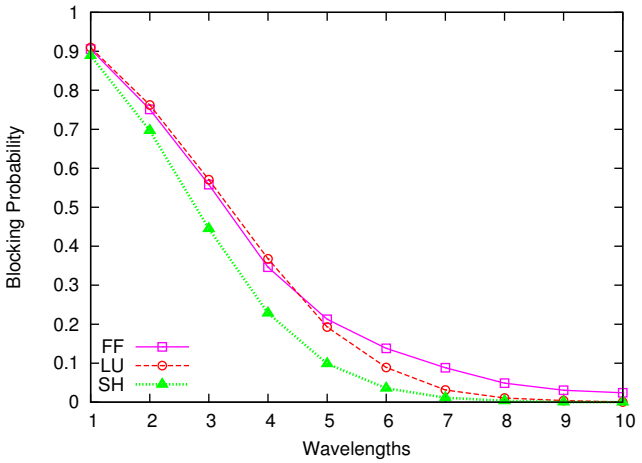
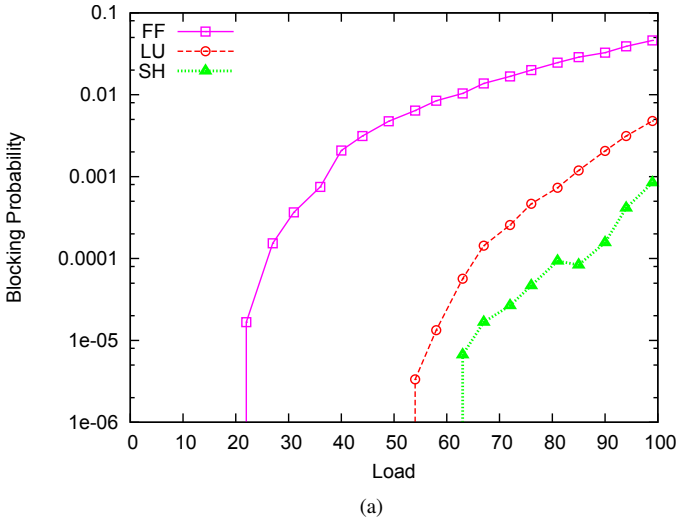


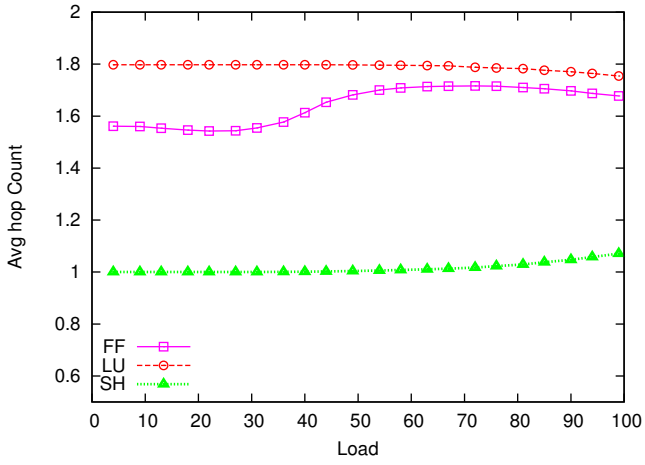
Fig. 4. Blocking performance for different number of wavelengths per link.

In Fig. 3 we show the blocking performance of the three heuristics (FF, LU and SH) for varying number of resources allocated per grid node. Similarly in Fig. 4, we depict the blocking performance for varying number of wavelengths per link. It can be observed from Fig. 3, that for number of resources less than 16 the *FF*, *LU*, and *SH* heuristics perform identically. Hence, we use 20 grid resources per node and 8 wavelengths per link for our performance evaluation.

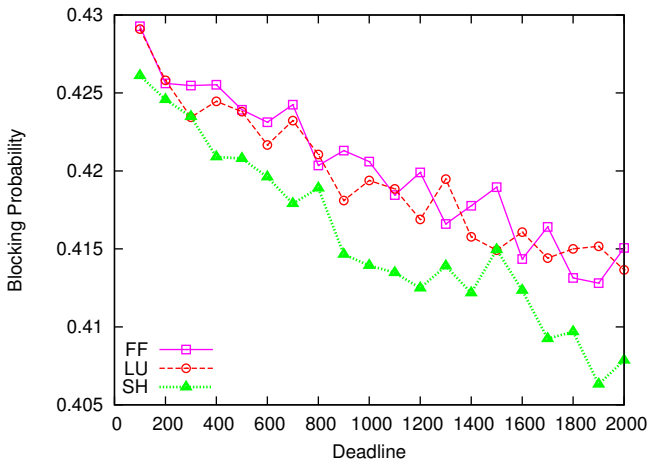
In Fig. 5(a) we compare job blocking probabilities for the FF, SH, and LU heuristics. It can be observed that the SH heuristic clearly out-performs the FF heuristic (by over two orders of magnitude at higher loads) and the LU heuristic (by almost an order of magnitude). This can be explained as follows: the least used heuristic selects the grid node (with sufficient number of available resources) depending upon the current resource usage of the nodes in the network. Thus



(a)



(b)



(c)

Fig. 5. Comparison of FF, SH and LU for (a) Average Blocking Probability, (b) Average Hop Count, (c) Deadline Performance.

it has a tendency to select grid nodes that are further away from the client node. This results in wavelength reservations on all the physical links connecting the client node to the particular (chosen) resource node. These reservations cause the future tasks to be blocked at network level. The *FF* heuristic performs the worst as it does not utilize any intelligent node selection criterion (grid/network usage). Rather it selects the smallest indexed node (with sufficient resources) to provision the request. Note also that since the *SH* heuristic chooses the shortest hop node to schedule a task, it results in lower average number of hops as compared to *LU* and *FF* heuristics as shown in Fig. 5(b). In Fig. 5(c) we compare the performance of the three heuristics for varying time-deadlines. As expected, we can observe an improvement in the blocking probability as the deadlines are increased for jobs. The tasks now have more time, wherein they can be pushed into future to get resources allocated to them.

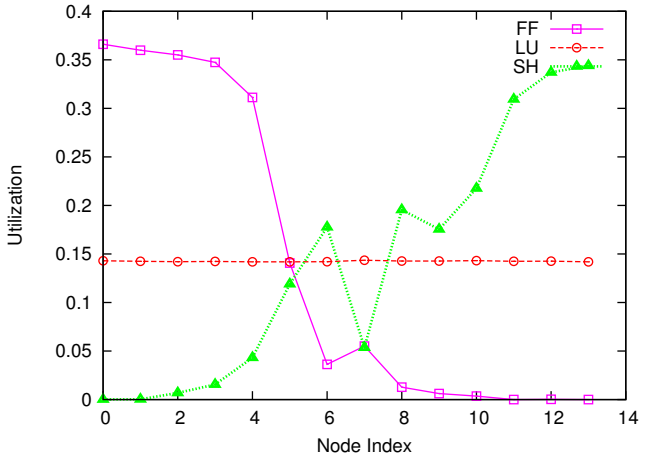


Fig. 6. Average Utilization across all the nodes

Finally in Fig. 6, we represent the utilization (defined as the the ratio of resource busy time to the total time duration of all the completed jobs) for the three heuristics. It can be observed that the LU heuristic has a constant utilization as it selects the least used node and attempts to utilize all the nodes in the network. There is very high utilization for the lower indexed nodes in FF, as it sorts the anycast destination set by the index numbers of the nodes and selects the node with the lowest index (and available resource) to provision a particular grid request. Note that the SH heuristic has higher utilization for higher indexed nodes, as in case of a tie (in terms of hop distance) between two nodes in the anycast destination set, we select the node with the higher index to provision the grid request.

VI. Conclusion

In this paper we addressed the problem of deadline-aware co-scheduling in lambda-grids (DACL) for advanced reservation requests. We proposed the anycast communication paradigm to solve the DACL problem and showed the benefits rendered by *co-anycasting* as opposed *grid-anycast*. We proposed three different anycast heuristics: *first free (FF), least used (LU)*, and *shortest-hop (SH)* to solve the DACL problem and evaluated their performance in terms of blocking, time-deadline, average hop count and utilization. It was observed that the SH heuristic out-performed the FF and LU heuristic. It was observed that the deadline plays an important role in improving the blocking performance. The primary limitation of this study is that of considering a homogeneous network. With most networks being heterogeneous (with different amounts of bandwidth on different links, considering resources, such as storage and scientific instruments.), the proposed algorithms can be extended for such networks. The proposed DACL problem can also be mathematically formulated and solved using integer linear programming (ILP). Dynamic routing techniques could also be employed to reduce network blocking. These are some of the potential areas we plan to investigate in our future work.

Acknowledgment

This work has been supported by the Department of Energy (DOE) COMMON project under grant DE-SC0004909 and the NSF CARGONET project under grant CNS-1218973.

References

[1] I. Chlamtac, A. Ganz, and G. Karmi, "Lightpath communications: an approach to high bandwidth optical WANs," *IEEE Trans. Commun.*, vol. 40, no. 7, pp. 1171–1182, July. 1992.

[2] Y. Wang, Y. Jin, W. Guo, W. Sun, W. Hu, and M. Wu, "Joint scheduling for optical grid applications," *J. of Optical Networking*, vol. 6, pp. 304–318, March 2007.

[3] H. Zang, J. P. Jue, and B. Mukherjee, "A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks." *SPIE Optical Networks Magazine*, vol. 1, no. 1, pp. 47–60, Oct 2000.

[4] Q. She, X. Huang, N. Kannasoot, Q. Zhang, and J. P. Jue, "Multi-resource manycast over optical burst-switched networks," in *Proc. ICCCN*, Aug. 2007, pp. 222–227.

[5] V. Lakshmiraman and B. Ramamurthy, "Joint computing and network resource scheduling in a lambda grid network," in *Proc. ICC*, May 2009.

[6] A. Ravula and B. Ramamurthy, "A tabu search approach for joint scheduling of resources in a lambda grid network," in *Proc. Globecom*, May 2010.

[7] N. Kannasoot, A. Patel, and J. Jue, "Sequential task anycast scheduling in optical burst switched networks," in *Proc. ICC*, May 2010.

[8] L. Kejie, T. Zhang, and A. Jafari, "An anycast routing scheme for supporting emerging grid computing applications in OBS networks," in *Proc. ICC*, June 2007.

[9] C. Castillo, G. Rouskas, and K. Harfoush, "On the design of online scheduling algorithms for advance reservations and QoS in grids," in *Proc. 21st IEEE International Parallel and Distributed Processing Symposium*, 2007.

[10] H. Nguyen, M. Gurusamy, and L. Zhou, "Provisioning lightpaths and computing resources for scheduled grid demands with location transparancy," in *Proc. OFC*, Feb 2008.

[11] G. Zervas, "Phosphorus grid-enabled GMPLS control plane (GMPLS): architectures, services, and interfaces," *IEEE Commun. Mag*, vol. 46, no. 6, pp. 128–137, June 2008.

[12] L. Battestilli, "Enlightened computing: An architecture for co-allocating network, compute, and other grid resources for high-end applications," in *Proc. International Symposium on High Capacity Optical Networks and Enabling Technologies*, Nov. 2007, pp. 18–20.

[13] A. Takefusai, "G-lambda: Coordination of a grid scheduler and lambda path service over GMPLS," in *Proc. Future Generation Computer Systems*, 2006.