

Cloud Computing over Metropolitan Area WDM Networks: The Light-trails Approach

Prasad Gokhale, Rajneesh Kumar, Tamal Das, Ashwin Gumaste

Department of CSE, Indian Institute of Technology, Bombay, India.

Email: {pkgokhale, rajneesh, tamaldas}@cse.iitb.ac.in, ashwing@ieee.org

Abstract: Cloud computing and IT-service provisioning is critical for the growth of enterprises in being able to provision computationally intensive applications. A high-speed networks infrastructure is necessary for the proliferation of cloud computing to meet disparate IT application demands. Light-trails – a generalization of lightpath with ability to provision sub-wavelength demands, meet dynamic bandwidth needs and cater to optical multicasting in a low-cost platform are investigated as a candidate technology for cloud computing. A time-slotted light-trail system is assumed and an algorithm is proposed based on utility concepts. Scheduling connections over light-trails in a timely manner to meet the tasks of an IT-service are considered. Memory resource management as a constraint is further incorporated in the algorithm thereby making the IT application pragmatically reside over the light-trails infrastructure. An exhaustive simulations study showcases the benefits of light-trails for cloud computing – the results obtained are over a wide range of services with serial, parallel and mixed set of constituent tasks.

Keywords: Light-trails, IT-virtualization, cloud computing.

I. INTRODUCTION

Cloud computing brings together complex Information Technology (IT) applications as a service on a shared networking infrastructure. This can lead to potential benefits such as significant cost reduction and performance enhancement. The underlying network is typically based on high-speed technologies and it needs to meet the needs of overlaid applications as described in [1], such as dynamic bandwidth allocation, efficient grooming and using a low-cost networking technology platform. Typically cloud computing applications would run over a metropolitan or regional network which would be based on an optical networking paradigm. However, contemporary optical technologies such as SONET/SDH, Gigabit Ethernet, RPR are either rigid in terms of bandwidth provisioning or are simply expensive to meet the disparate and diverse needs of cloud computing. Light-trails [2] were proposed as a generalization of a lightpath and are efficient high-speed networking solutions that have the ability to meet cloud computing applications [3]. Light-trails have been demonstrated [4] for features such as dynamic provisioning [5], sub-wavelength grooming [6] and optical multicasting [7] – all essential features for cloud computing. We present light-trails as a case technology for IT-over optical networks and in particular for cloud computing. To this end, an algorithm that binds IT-applications to a light-trail based WDM metro network is proposed, analyzed and simulated. This algorithm is based on a two-stage valuations technique and makes good use of the advantages of wavelength sharing in the optical domain while achieving fast response – much desired by IT applications. Simulation results validate the use of our proposed algorithm and thereby bring out the benefits of light-

trails as an infrastructural solution for complex IT applications in the domain of cloud computing. This paper is an extension of [3], leading to a new algorithm for cloud applications with pragmatic assumptions such as time-slotted system, and has as its objective – minimization of network resources. The work presented in this paper is also motivated from [11] in which an LP is proposed for the case of static scheduling. Our work leads to a heuristic pragmatic algorithm for dynamic service needs.

The paper is organized as follows: Section II is a primer on light-trails. In Section III, we describe the basic network model that is instructive in defining the IT over optical networks topology that leads to cloud computing. Section IV details the algorithm for scheduling IT-services over light-trails. Different aspects of the algorithm are presented such as growth and scheduling. Section V describes the simulation model for evaluating light-trails as well as the results from simulations, while Section VI concludes the paper.

II. LIGHT-TRAILS

Light-trails are defined as a generalization of a lightpath such that multiple nodes can take part in communication along the path [8]. A light-trail is analogous to a wavelength bus, thus exhibiting functionalities of optical multicasting and being able to provide time-differentiated sub-wavelength communication to nodes in the light-trail. A light-trail based network leads to spatial wavelength grooming and enables dynamic bandwidth provisioning. Nodes in a light-trail have properties to support bus functionality. In particular, nodes support, on a per-wavelength basis drop-and-continue and passive add properties as well as an out-of-band control channel that enables arbitration of communication within the bus. The OOB control channel is optically dropped and electronically processed at every node. A sample node architecture using Wavelength Selectable Switches (WSS) as the demux/mux is shown in Fig. 1. Also shown in Fig. 1 is a comparison of light-trails to lightpaths. Signal flow in a light-trail is regulated at either ends of the bus by the extreme nodes called the convener node and the end-node respectively. Signal flow occurs in a unidirectional manner – from the convener to the end-node. When a node communicates to another node over a light-trail, it does so by forming a *connection*. Connections are provisioned over the wavelength bus without any optical switch configuration. When a node communicates over the light-trail, other nodes queue their data in electronic buffers. Buffers are implemented in the electronic domain, and these store data, while other nodes use the light-trail. A light-trail MAC protocol is essential to regulate communication. The light-trail is

assumed to be time-slotted and nodes compete for time-slots depending on their service requirements. A centralized arbiter is responsible for communication within the light-trail and grants time-slots to nodes depending on their service requests. Since multiple nodes all-optically share the wavelength bus, they individually are allocated sub-wavelength granularity, and this type of grooming is defined as spatial grooming due to the geographic (or spatial) diversity of nodes time-sharing the bus. To support a bus, nodes on a per-wavelength basis have two optical couplers – in 1x2 and 2x1 configurations (called drop and add couplers) – that act as power splitter and power combiner respectively. An ON/OFF switch separates the two couplers and is ON at all the intermediate nodes, while it is in the OFF position at the convener and the end-node. The state of the switch is not changed for communication and changes only when a light-trail is setup, torn down or dimensioned [9]. From a cloud computing perspective, we desire to compute when to set up light-trails, once a light-trail is set up how to assign connections to requests within the light-trail and how to control the virtual topology across multiple interconnected ring WDM networks.

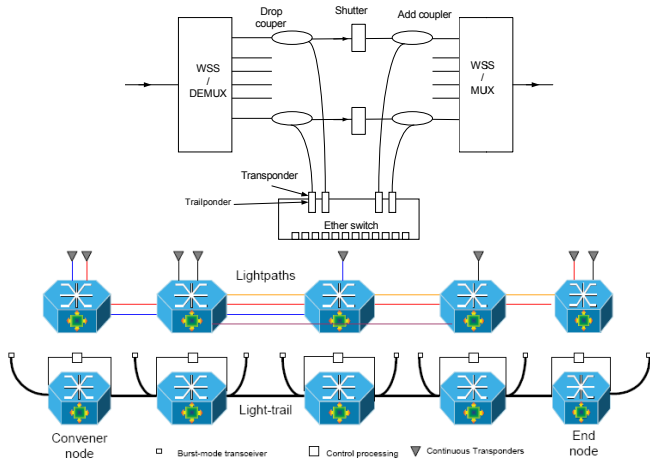


Fig. 1. Light-trail node architecture (above) and lightpath comparison (below).

III. NETWORK MODEL

Notations: We define the network topology by an undirected graph $G(N,L)$, where N is the set of nodes and L the set of fiber-links. Denote the i^{th} node by N_i with processing power $p(N_i)$, equipped with memory capacity $m(N_i)$. Each node has a local service buffer that holds the services which arrive at *this* node. The capacity of each link is assumed to be constant and is denoted by C . The universal set of services that the network can provision is given by $S=\{S_1, S_2, \dots, S_n\}$. A service S_i is associated with a bandwidth requirement of $bw(S_i)$ and is divisible into several interdependent and atomic tasks. We assume that the bandwidth requirement of each constituent task of a service is the same. The node at which the service enters the network is denoted by $g(S_i)$. We denote the universal set of tasks by $T=\{T_1, T_2, \dots, T_m\}$. Task T_i requires $p(T_i)$ units of processing power, $m(T_i)$ units of memory resource and is subject to delay bound $\delta(T_i)$. The delay bound of a service is the cumulative time duration of the critical path in its *task*

dependency graph (TDG) (Fig. 3). Further the start and end-times of T_i 's execution are denoted by $t_s(T_i)$ and $t_e(T_i)$, respectively. To provision a given task at a compatible node, we first check if the memory requirement of the task is satisfied by the node followed by its processing power requirements, since in case of low processing power at a node, a task can still be processed over a long time, but in case of insufficient memory resource, the task *cannot* be processed at the node. Of all such compatible nodes, we choose the one with *best fit* for the task's memory requirement. Further, the task currently being executed at node N_i is denoted by $ct(N_i)$, and the node at which task T_i is currently being processed is denoted by $parentNode(T_i)$. The task dependency graph of service S_i is denoted as the directed graph $G_i = \{\bar{T}_i, \bar{E}_i\}$, where \bar{T}_i denotes

the set of tasks that together constitutes S_i and \bar{E}_i denotes the set of directed links between interdependent tasks. $E(T_i, T_j)$ indicates the directed edge from T_i to T_j . This implies that T_j cannot be executed before it receives the data from T_i . Each such edge $E_i \in \bar{E}_i$ is associated with the amount of data $\rho(E_i)$ that needs to be transferred between its interconnecting tasks.

Fig. 2 and Fig. 3 illustrate the above mentioned convention. In Fig. 2, each of the nodes N_1, \dots, N_5 are labeled with a 2-tuple, where the first parameter (p_i) denotes the processing power of the node and the second parameter (m_i) denotes its memory capacity. Further, each of the edges is labeled with their respective bandwidth (C). Fig. 3 displays the task dependency graph for a service. Each of the tasks T_1, \dots, T_5 is associated with a 3-tuple, *viz.*, processing power (p_i), memory requirement (m_i) and its delay bound (δ_i). The amount of data (ρ_i) that needs to be transferred between successive tasks is labeled on the respective edges of the task dependency graph.

Problem Statement: The *input* to our problem is the global set of services existing at time t awaiting resources – denoted by $\Omega(t)$ in a time-slotted system. The *output* is an *efficient schedule* of $\Omega(t)$ that tries to optimize total execution time while using a *heuristic minimum number of light-trails*, subject to (1) the capacity constraints of the links and (2) the resource requirement and delay bound of each service. We consider a heuristic algorithm as proposed in the next Section, as opposed to an LP formulation (for example as in [11]) due to the LP being NP-complete (analogous to bin-packing). Further, the LP in [11] and other work in [7, 9] is stochastically agnostic, meaning demands are all known ahead in time. We will consider the case where demands are not known ahead in time, thus making our model pragmatic.

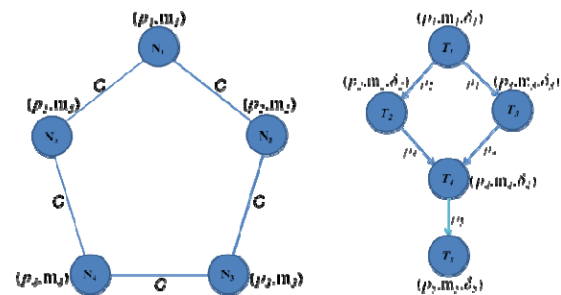


Fig. 2. Sample Network topology Fig. 3. Task Dependency Graph.

IV. CLOUD COMPUTING OVER LIGHT-TRAIL ALGORITHM

Our approach is to prioritize the incoming service requests motivated from the scheduling mechanism presented in [9]. The idea is that a service which will be the first to be dropped, if not provisioned, is assigned the highest priority. Further we adapt the virtual light-trail topology growth algorithm [9] for efficient and dynamic data transfer between interdependent tasks. As a light-trail is a shared, unidirectional, wavelength bus, multiple connections need to be provisioned, subject to incoming requests. To achieve this, we consider a time-slotted system. The time-slots are assigned to a node based on its utility for the same. The utility is calculated based on the node's buffer occupancy and delay stringency. *Algorithm 1* provides the integrated schedule for the problem under consideration. It runs in parallel with *Algorithm 3* and *Algorithm 4*. *Algorithm 5* provides the virtual topology growth for provisioning an incoming connection request and *Algorithm 6* provides the mechanism to grant permission for data transfer on the light-trail between competing connection requests. A detailed description of our algorithm is now presented.

Algorithm Conventions: We now discuss the conventions used for our analysis. $K(t)$ denotes the set of established light-trails at time t . f_{ij} is a flow from node N_i to node N_j in the network. t_{HP} is the time required to setup a light-trail, whereas T_S denotes the duration of each time-slot. We denote a generic light-trail by k or a particular i^{th} light-trail by k_i . $\|N_i \oplus N_j\|_k = 1$, if N_i is upstream of N_j on light-trail k , and 0 otherwise. U_k denotes the utility (total flow divided by the capacity) of light-trail k . The upper and lower bound on the utility is denoted by UB_{Thresh} and LB_{Thresh} , respectively. The start- and end-nodes of light-trail k is denoted by $conv(k)$ and $end(k)$. Light-trail k is established on wavelength λ_k . $\psi_{ik}(t)$ denotes the maximum duration of time that N_i can wait for a transmission slot after which its most stringent service packet will be dropped, which in turn results in its parent service being dropped.

Algorithm 1 (Integrated Global Scheduler): This algorithm outlines the steps required to efficiently provision an incoming set of services and outputs an efficient schedule for its provisioning over the network. Its pseudo-code is stated in Listing-1.

Each node checks for the services in its local service buffer to provision it using *only* its own resources. If it is not possible to schedule the service, then the node enqueues such a service at the global service queue. In the global service queue, services are prioritized as per *Algorithm 2* and are provisioned in order of decreasing priority. For a service waiting in the global service queue, the scheduler provisions the constituent tasks of a service in the sequence of their execution based on the service's TDG. The scheduler always aims to minimize the inter-node communication in the network. Thus a task is allocated a node so as to minimize the data exchange from its predecessor tasks using light-trails.

For an *entry task* in a TDG, which is the first task to be executed for a service, the scheduler checks if it can be provisioned at (1) a node (say N_{parent}) at which its parent service arrived, or (2) a node in the same ring to which N_{parent} belongs, or (3) a node in the ring neighboring to N_{parent} , or (4) any node in the network in the decreasing order of its proximity to N_{parent} , in that order. When a task is provisioned at a node, its busy schedule, which is the set of time-slots in which a node is reserved by tasks for their execution, is updated. Further, a connection request from N_{parent} to the node under consideration is sought using *Algorithm 5*.

For an intermediate task in a TDG, the scheduler iterates over all its predecessor task(s), and tries to provision it at one of its predecessor task's *parentNode*, in decreasing order of inter-task data exchange. If the concerned task cannot be provisioned at any such predecessor node, the scheduler checks if it can be provisioned at (1) a node in the same ring as the predecessor node from which maximum inter-task data transfer needs to take place (say $N_{maxDataTrans}$), or (2) a node in the ring neighboring to $N_{maxDataTrans}$, or (3) any node in the network in the decreasing order of its proximity to $N_{maxDataTrans}$, in that order. In the event of the task being provisioned at a node, the busy schedule of the corresponding node is updated, and a connection request from *parentNode* of each predecessor of this task to the selected node is sought using *Algorithm 5*.

Otherwise, if it cannot be provisioned at any of the above scenarios, then the task and its parent service are dropped from the network.

```

 $A_i(t) \leftarrow$  Local service buffer at node  $N_i$ 
for each  $N_i \in N$ 
  while  $A_i$  is not empty
     $S_0 \leftarrow head(A_i(t))$ 
    if  $S_0$  can be provisioned at  $N_i$ 
      Update the busy schedule of  $N_i$ 
    else
      Enqueue  $S_0$  in the global service queue  $\Omega(t)$ 
    end if
  end while
end for
 $\Omega(t) \leftarrow$  Prioritize  $\Omega(t)$  using Algorithm 2
while  $\Omega(t)$  is not empty
   $S_i \leftarrow$  Highest priority service of  $\Omega(t)$ 
   $G_i = \{\bar{T}_i, \bar{E}_i\} \leftarrow$  Task dependency graph for  $S_i$ 
  while  $\bar{T}_i$  is not empty
     $T_j \leftarrow head(\bar{T}_i)$ 
     $\Psi(T_j) \leftarrow$  Set of available nodes compatible with  $T_j$ 
    if  $T_j$  is an entry node in  $G_i$ 
      if  $T_j$  can be provisioned at  $g(S_i)$ 
        Assign  $T_j$  to  $g(S_i)$ 
        Update the busy schedule of  $g(S_i)$ 
      else if there exists  $N_i \in \Psi(T_j)$  and  $N_i$  belongs to the same ring as  $g(S_i)$ 
        Assign  $T_j$  to  $N_i$ 
        Update the busy schedule of  $N_i$ 
        Add a connection request from  $g(S_i)$  to  $N_i$  in the global event queue
      else if there exists a node  $N_i \in \Psi(T_j)$  and  $N_i$  belongs to any of the
        neighboring rings of  $g(S_i)$ 
        Assign  $T_j$  to  $N_i$ 
        Update the busy schedule of  $N_i$ 
        Add a connection request from  $g(S_i)$  to  $N_i$  in the global event queue
      else if  $T_j$  can be assigned to any node  $N_i \in \Psi(T_j)$ 
        Update the busy schedule of  $N_i$ 
    end while
  end while
end while

```

```

    Add a connection request from  $g(S_i)$  to  $N_i$  in the global event queue
    else drop  $S_i$  from  $\Omega(t)$ 
    end if
end if
else
     $PD(T_j) \leftarrow$  Set of all predecessor tasks of  $T_j$  sorted in descending order of
     $\rho(E(PD(T_j)), T_j)$ 
     $parentNode(T_k) \leftarrow$  Node at which  $T_k$  is executing
    for all  $N_j \in parentNode(PD(T_j))$ 
    if  $ct(N_j) \in PD(T_j)$ 
        Assign  $T_j$  to  $N_j$ 
        Update busy time of  $N_j$ 
         $PD(T_j) \leftarrow PD(T_j) / ct(N_j)$ 
    if  $PD(T_j)$  is not empty
        Add a connection request from  $parentNode(PD(T_j))$  to  $N_j$ 
    end if
    end for
    if  $T_j$  cannot be provisioned at any  $N_j \in parentNode(PD(T_j))$ 
    if  $T_j$  can be provisioned at any node ( $N_o$ ) from the same ring as the
         $parentNode(PD(T_j))$  with the maximum data transfer
        Assign  $T_j$  to  $N_o$ 
        Update the busy schedule of  $N_o$ 
        Add a connection request from  $parentNode(PD(T_j))$  to  $N_o$ 
    else if  $T_j$  can be provisioned at any node ( $N_o$ ) from the neighboring rings
        as the  $parentNode(PD(T_j))$  with the maximum data transfer
        Assign  $T_j$  to  $N_o$ 
        Update the busy schedule of  $N_o$ 
        Add a connection request from  $parentNode(PD(T_j))$  to  $N_o$ 
    else if  $T_j$  can be provisioned at any node ( $N_o$ ) in the network
        Assign  $T_j$  to  $N_o$ 
        Update the busy schedule of  $N_o$ 
        Add a connection request from  $parentNode(PD(T_j))$  to  $N_o$ 
    else Drop  $S_i$  from  $\Omega(t)$ 
    end if
    end if
end if
end while
return the schedule

```

Listing-1: Pseudo Code for Integrated Global Scheduler

Algorithm 2 (Service Prioritization Algorithm): In this algorithm, we compute the relative priority of different services in the global service queue. Priority of a service is determined based on the maximum amount of time that a service can wait before it will be dropped, normalized over its delay bound. The input to this algorithm is $\Omega(t)$ whereas the output is $\Omega(t)$ sorted in descending priority as shown in Listing-2.

```

    arrivalTime  $\leftarrow$  Arrival times of services in  $\Omega(t)$ 
     $\Delta \leftarrow$  Delay bounds of services in  $\Omega(t)$ 
     $j \leftarrow 1, t \leftarrow$  current time instant
    while ( $\Omega(t)$  is non-empty)
         $S_i \leftarrow \arg \min \{ [\Delta(i) - (t - arrivalTime(i))] / \Delta(i), \forall i \}$ 
        output( $j$ )  $\leftarrow S_i$ 
         $\Omega(t) \leftarrow \Omega(t) - S_i$ 
        Re-evaluate arrivalTime,  $\Delta$ 
         $j \leftarrow j + 1$ 
    end while
    return output

```

Listing-2: Pseudo Code for Service Prioritization Algorithm

Algorithm 3 (Task completion event handler): The task completion event handler upon completion of a task releases the resources assigned to it as depicted in Listing-3.

```

    if (a task has finished its execution and released its resources)
        Update the busy schedule of the  $parentNode$  of the finished task
    end if

```

```

    Reprioritize  $\Omega(t)$  using Algorithm 2
end if
Listing-3: Pseudo Code for Task completion event handler

```

Algorithm 4 (Service arrival event handler): On arrival of a new service request, it is queued at the local service buffer at the node at which it arrived. Then the scheduler (*Algorithm 1*) is called at the next time slot to provision it. The pseudo code to accomplish this is illustrated in Listing-4.

```

    if (new service request has arrived)
        Enqueue this service at the node at which it arrived
        Call the scheduler at the next time-slot
    end if
Listing-4: Pseudo Code for Service arrival event handler

```

Algorithm 5 (Connection Request Handler): For a task to begin its execution either it may be processed on the same node as its predecessor(s) or the processed data from all its predecessors should be transferred (before its execution) to the node on which it is provisioned. In the latter case, there must exist light-trail(s) connecting the ingress node(s) to the egress node. To provision a light-trail, we first check if an existing light-trail can accommodate the required connection. If not, whether any existing light-trail can be extended at either ends to accommodate both the required nodes. If not, then we create a new light-trail to provision the data transfer and inform the scheduler to provision the connection on this light-trail in subsequent future slots based on its utility for the same. The inputs to this algorithm are N_{src}, N_{dest}, G_i , whereas the output is a light-trail provisioned between N_{src} to N_{dest} in G_i (See Listing-5).

```

     $N_{src} \leftarrow$  Source node of the required connection
     $N_{dest} \leftarrow$  Destination node of the required connection
     $bw \leftarrow$  bandwidth requirement of the required connection
    if an existing light-trail ( $LT_p$ ) connects  $N_{src}$  and  $N_{dest}$  with  $bw$  units of free
    bandwidth
        then assign the connection to  $LT_p$  between  $N_{src}$  and  $N_{dest}$ 
    else if an existing light-trail ( $LT_p$ ) includes  $N_{src}$  (or  $N_{dest}$ ) and can be
    extended to  $N_{dest}$  (or  $N_{src}$ )
        Extend  $LT_p$  to  $N_{dest}$  (or  $N_{src}$ ) and assign the required connection
    else
        Create a new light-trail between  $N_{src}$  and  $N_{dest}$  and assign connection
    end if
    Update  $L$ 
    Add an event to schedule this connection at  $LT_p$ 

```

Listing-5: Pseudo Code for Light-trail Provisioning

Algorithm 6 (Granting Light-trail access): To provision multiple connection requests on a light-trail, we deploy Time Division Multiplexing (TDM). To grant light-trail access privileges to a node, we prioritize the nodes based on its utility for a given time-slot. Each node has w incoming buffers and w outgoing buffers, where w is number of wavelengths on the fiber. Each buffer is associated with a particular wavelength. Incoming buffer enqueues packets to be sent using light-trail provisioned on its particular wavelength. Each light-trail has one member node as the arbiter that decides the winning node. The arbiter node gets the utility requirement from each node that wants to transmit data. The node computes utility factor as maximum of buffer occupancy and the amount of time task running at egress node can wait before it is dropped which is normalized over the deadline of the task. The arbiter then grants

access to a node whose utility value is the highest. The input to the algorithm is a light-trial and the output is a node that will transfer the data in next time slot.

```

k ← Light-trail
utility ← 0
t ← Current time instant
Bik(t) ← Buffer occupancy at node Ni for light-trail k at time t
Bmax ← Maximum size of the buffer.
egress(Ni) ← Node to which Ni wants to send data
member(k) ← Set of all member nodes for light-trail k
while(member(k) is not null)
    Ni ← head(member(k))
    utility(Ni) ← max {  $\frac{B_{ik}(t)}{B_{max}}$ ,  $\left(1 + \frac{t_e(ct(egress(N_i)) - t)}{t_e(ct(egress(N_i)) - t_e(ct(N_i)))}\right)^{-1}$  }
    winningNode ← argmax {utility(Ni)}
end while
return winningNode
    
```

Listing-6: Pseudo Code to grant Light-trail access

V. SIMULATION AND NUMERICAL RESULTS

A discrete event simulator was developed in JavaTM for cloud environment with under laid light-trail network. It was designed and simulated over a WDM network whose topology is as shown in Fig. 4. The network consists of 5 interconnected rings and a total of 90 nodes, with each ring consisting of 14-21 nodes and 40 wavelengths. Each time-slot is 1ms in duration. Each pair of neighboring rings is interconnected with 2 ROADMs with the architecture of the nodes shown in Fig. 1.

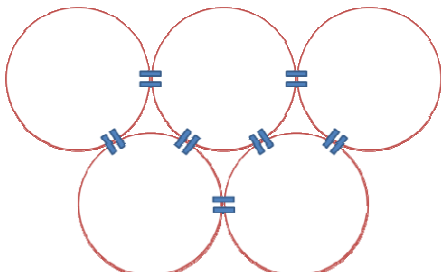


Fig. 4. Network Topology

We consider 3 classes of services – *serial*, *parallel* and *mixed*. The aim of considering this variation is to be able to measure light-trail performance for all extreme class of services. A service can potentially be broken into tasks.

Serial services are those for which no two constituent tasks can be executed at the same time, whereas in case of parallel services multiple constituent tasks can be executed simultaneously as shown in Fig. 5-6. Mixed services are a hybrid of the above two service types, with certain tasks that *must be* executed in serial order, while the rest that can be executed in parallel order as shown in Fig. 7. For each of these 3 service types, we consider 3 different instances, having 4 to 10 tasks each in their TDG.

Service arrival at each node follows Poisson distribution with arrival rates ranging from 10-100 services/sec and predefined holding times. Light-trail line rate was assumed to be 1Gbps. Network topology and TDGs were manually given as input to the simulator. Each service had a predefined

execution deadline (inclusive of the processing time) between 130 and 300ms. Load was measured as sum of all the traffic across the network.

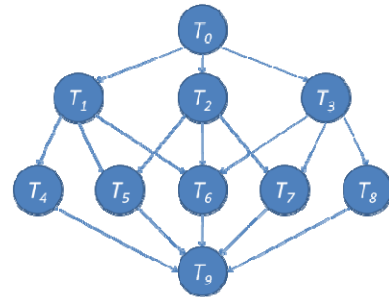


Fig. 5. Sample TDG for a Parallel service class

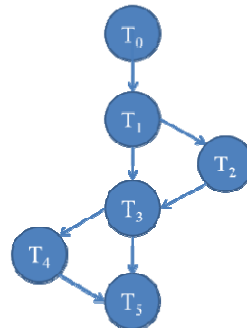


Fig. 6. Sample TDG for a Serial service class

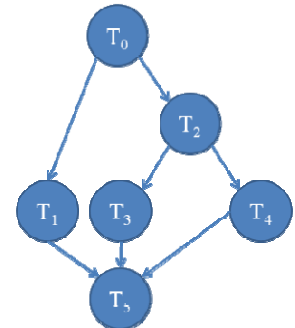


Fig. 7. Sample TDG for Mixed service class

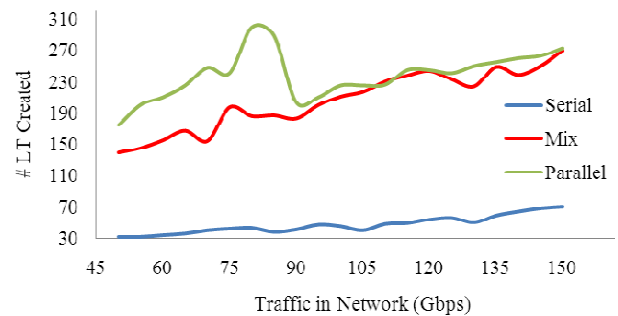


Fig. 8. Number of Light-trails created versus Network traffic

Shown in Fig. 8 is a viewgraph that describes the number of light-trails created by our algorithm as a function of ingress load (calculated in Gbps). To plot the viewgraph we consider the most-parallel (or those services with the maximum amount of concurrency) and the most serial (or those services with the least amount of concurrency). The serial services require the least number of light-trails – as the traffic is somewhat static – which also implies that the communication component is significantly offset by the computation component [10]. The parallel scheme is unpredictable in terms of the number of light-trails and shows sudden increases – especially at medium loads. This is possibly due to the parallelism in the services implying that light-trails are created exclusively to meet this parallelism. The mixed services profile requires almost monotonically increasing number of light-trails as a function of

load. This scheme shows us that there is some merit in not having exclusively serial or exclusively parallel task profiles.

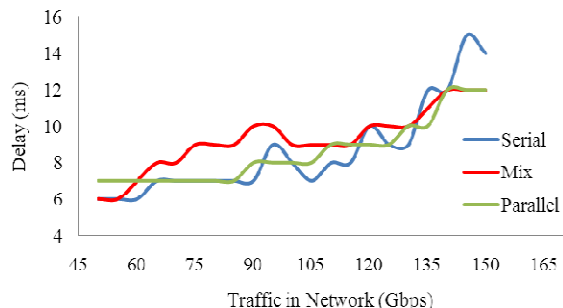


Fig. 9. Average Delay versus Network traffic

Shown in Fig. 9 is the viewgraph that shows the variation of average delay experienced by a service versus the ingress load. The algorithm is tuned to ensure that the delay is within service limits – hence the serial, parallel and mixed schemes perform very close to each other, except at very high-loads, where the serial scheme performance degrades. At high loads, the serial scheme drops data and hence average delay improves – though at the price of blocked services. This shows an upper bound for the assignments of tasks serially in a cloud-like environment.

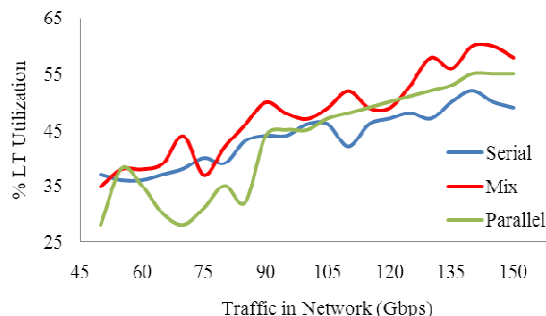


Fig. 10. Average light-trail utilization versus Network traffic

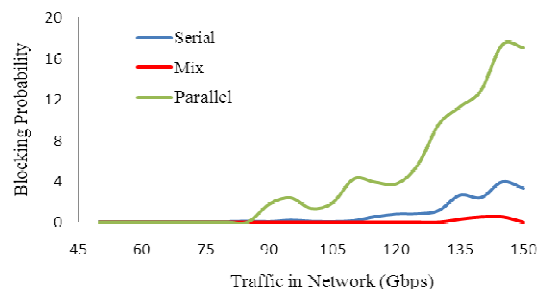


Fig. 11. Blocking Probability versus Network traffic

Shown in Fig. 10 is the viewgraph that shows the relationship between average utilization of a light-trail versus the ingress load. In the mixed case, parallelism ensures better utilization, while serialism enables enough time to configure the light-trails thereby achieving higher utilization. Shown in Fig. 11 is a viewgraph that shows the relationship between blocking probability of a service versus the ingress load. The parallel services are a challenge for provisioning due to the quick need to provision connections over light-trails and this is evident from the high blocking experienced (almost 14% at full load).

If we would have relaxed the delay requirements, then the parallel services would not have experienced large blocking. The mix service has the best case blocking. The simulation results give a good insight as to how to locate tasks, and the key result is to maintain a good balance between serial and parallel tasks.

VI. CONCLUSION

An algorithm for provisioning complex IT applications over optical networks is presented. This algorithm is tested over a light-trail based WDM metro network. Complex IT processes that can be reduced to disparate tasks are considered, with each task being able to be processed at geographically diverse set of nodes. The algorithm works to minimize the latency in processes by aiming to parallelize the algorithm with distributed processing over a set of nodes. The light-trail paradigm is well-investigated to meet the needs of IT services. Utilization, latency, delay and virtual topology characteristics are measured over an intense simulations exercise thereby showcasing light-trails as an efficient solution for cloud computing.

Acknowledgement: This work is funded by the European Commission Project Geysers [1] and the Ministry of Information Technology, India.

References:

- [1] Online Reference: <http://www.geysers.eu/>
- [2] A. Gumaste and S. Q. Zheng, "Optical Storage Area Networks: The Light-trails Approach," in IEEE Communications Magazine March 2005 pp. 72-78 Vol. 21. No. 3.
- [3] P. Gokhale, T. Das and A. Gumaste, "Cloud Computing over Light-trail WDM Core Networks", IEEE/OSA Optical Fiber Commun Conf. 2010, San Diego.
- [4] A. Gumaste, N. Ghani P. Bafna, A. Lodha, A. Agrawal, T. Das and S. Zheng, "DynaSPOT: Dynamic Services Provisioned Optical Transport Test-bed - Achieving Multi-Rate Multi-Service Dynamic Provisioning using Strongly connected Light-trail (SLiT) Technology," in IEEE Journal of Lightwave Technology, Jan 2008.
- [5] A. Gumaste, P. Palacharla and T. Naito, "Performance Evaluation and Demonstration of Light-trails in Shared Wavelength Optical Networks (SWON)", 31st European Conf. On Optical Communication (ECOC) 2005.
- [6] A. Gumaste, A. Jukan, A. Lodha, C. Xue and N. Ghani, "A Novel Node Architecture for Light-trail Provisioning in Mesh WDM Metro Networks," in IEEE/OSA 24th Optic Fiber Communications (OFC) conference 2008.
- [7] S. Balasubramanian and A. K. Somani, "A Comparative Study of Path Level Traffic Grooming Strategies for WDM Optical Networks with Dynamic Traffic," ICCCN 2008 Virgin Islands, Aug. 2008
- [8] A. Gumaste and I. Chlamtac, "Light-trails: A Novel Conceptual Framework for Conducting Optical Communications", 3rd IEEE Workshop on High Performance Switching and Routing, Proceedings of HPSR, 2003
- [9] A. Gumaste and T. Das, "Two Stage Algorithm for Light-trail Topology Growth," (Under Submission)IEEE/OSA Journal of Optical Communication Networks 2010.
- [10] Viswanathan, S., Veeravalli, B., and Robertazzi, T.G., "Resource Aware Distributed Scheduling Strategies for Large-Scale Computational Cluster/Grid Systems," IEEE Transactions on Parallel and Distributed Systems, Vol. 18, No. 10, Oct. 2007, pp. 1450-1461.
- [11] X. Luo and B. Wang, "Integrated Scheduling of Grid Applications in WDM Optical Light-Trail Networks"IEEE/OSA Journal of Lightwave Technology, Vol. 27, Issue 12, pp. 1785-1795