Resource Allocation in Slotted Optical Data Center Networks

K. Kontodimas[1], K. Christodoulopoulos[1], E. Zahavi[3], E. Varvarigos[1,2]
[1]School of Electrical and Computer Engineering, National Technical University of Athens, Greece
[2]Department of Electrical and Computer Systems Engineering, Monash University, Australia
[3]Mellanox Technologies Ltd., Yokneam, Israel
{kontodimas, kchristo}@mail.ntua.gr, eitan@mellanox.com, vmanos@central.ntua.gr

Abstract— **The introduction of all-optical switching in data center interconnection networks (DCN) is key for addressing several of the shortcomings of state-of-the-art electronic switched solutions. Limitations in the port count and reconfiguration speed of optical switches, however, require novel DCN designs offering network scalability and dynamicity. We present the NEPHELE DCN which relies on hybrid electro-optical top-of rack (TOR) switches to interconnect servers over multi-wavelength all-optical rings. We described in detail the NEPHELE control cycle which follows the SDN paradigm. We evaluate the performance of NEPHELE regarding the effect of the control plane delay under realistic traffic.**

Keywords— Time-Wavelength-Space division multiplexing; slotted and synchronous operation; dynamic resource allocation, scheduling; matrix decomposition

I. INTRODUCTION

The widespread availability of cloud applications to billions of end-users and the emergence of platform- and infrastructure-as-a-service models rely on concentrated computing infrastructures, the Data Centers (DCs). As traffic within the DC (east-west) is higher than incoming/outgoing traffic, and both are expected to continue to increase [1], DC networks (DCN) play a crucial role. High throughput, scalable and energy/cost efficient DCN networks are required to fully harness the DC potential.

State-of-the-art DCNs are based on electronic switches connected in fat-tree topologies using optical fibers, with electro-opto-electrical transformation at each hop [2]. However, fat-trees tend to underutilize resources, require a large number of cables and switches, suffer from poor scalability and upgradability (lack of transparency), and they result in very high energy consumption [3], [4]. Application driven networking [5], [6], an emerging trend, would benefit from a network that flexibly allocates capacity where needed.

The introduction of optical switching in DCN is a key for solving these shortcomings. Many recent works proposed hybrid electrical/optical DCN, a survey of which is presented in [7]. The authors of [8] and [9], proposed a DCN in which heavy long-lived (elephant) flows are selectively routed over an optical circuit switched (OCS) network, while the rest of traffic goes through the electronic packet switched (EPS) network. The identification of elephant flows is rather difficult on the fly, while it was observed that such flows are not very typical [4], making it difficult to sustain high OCS utilization. Instead, a high connectivity degree is needed [4]. To enable higher connectivity, [10] proposed and prototyped a very dense hybrid DCN that also supports multi-hop connections. The total delay, including control plane and OCS hardware reconfiguration (micro electro-mechanical system – MEMS – switches), was measured to be in the order of hundreds of msec. Multi-hop routing was exploited as *shared* circuits in [11] controlled via extended OpenFlow [12], showing that circuit sharing compensates for slow OCS reconfigurations.

Other proposed DC interconnects completely lack electrical switches. Proteus, an all-optical DCN architecture based on a combination of wavelength selective switches (WSS) and MEMS was presented in [13]. Again, multi-hop is used to improve the utilization and hide the low reconfiguration speed of MEMS. [14] introduced hybrid OCS and optical packet/burst switching (OPS/OBS) architectures, controlled using SDN. Various other architectures based on OPS/OBS were proposed in [7], [15] and references therein. However, OPS/OBS technologies are not yet mature, so their current target could be only small-scale networks with limited upgradability potentials.

The authors in [16] presented a hybrid DCN architecture called Mordia, which uses WSS to achieve switching times of 11.5 μs. Mordia operates in a dynamic slotted manner to achieve high connectivity. However, the scalability of Mordia is limited as it uses a single wavelength division multiplexing (WDM) ring that can support traffic for a few racks, while resource allocation algorithms exhibit high complexity and cannot scale to large DCs.

The European project NEPHELE developed an optical DCN that leverages hybrid electrical/optical switching with SDN control to overcome current datacenter limitations [17]. To enable dynamic and efficient sharing of optical resources and collision-free communication, NEPHELE operates in a synchronous slotted manner. Timeslots are used for rack-to-rack

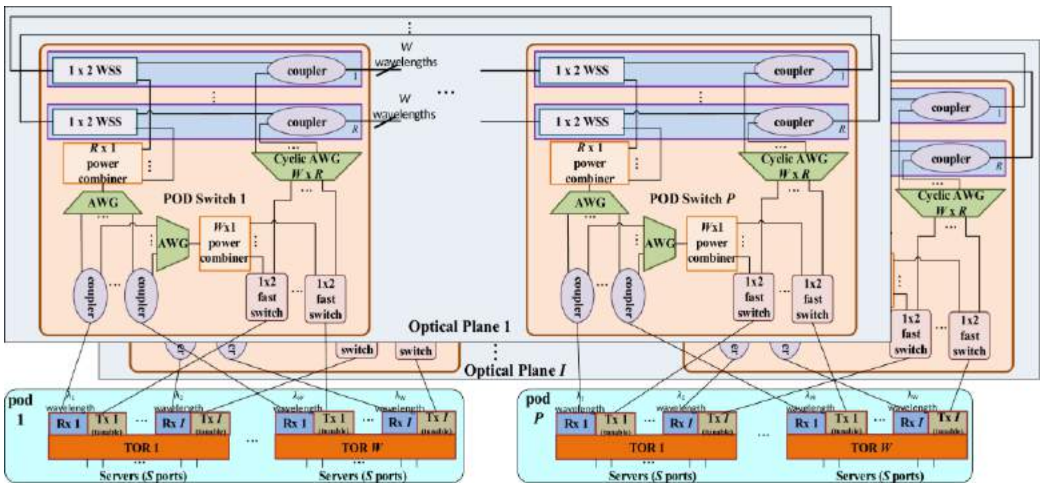

Fig. 1. NEPHELE Resource allocation and Data cycles.

communication and are assigned dynamically, on a demand basis, so as to attain efficient utilization, leading to both energy and cost savings. Moreover, multiple wavelengths and optical planes are utilized for a scalable and high capacity DC network.

The NEPHELE network relies on WSSs, which are faster than MEMS used in [8]-[11], and more mature than OPS/OBS used in [14]-[15]. The fast switching times, along with the dynamic slotted operation, provide high and flexible connectivity. Compared to Mordia [16], which also relies on WSS, NEPHELE is more scalable: it consists of multiple WDM rings, re-uses wavelengths, and utilizes cheap passive routing components and scalable scheduling schemes. We present fast scheduling algorithms for NEPHELE DCN to meet dynamic reconfiguration requirements and evaluate their effect along with the control plane overhead on the performance of realistic applications.

In the following we shortly present the NEPHELE data plane (Section II) and then we describe in detail its control cycle (Section III) and the scheduling approaches (Section IV). Finally, using a packet level simulator we evaluate the effect of the control plane delay under realistic traffic (Section V).

II. NEPHELE Network Architecture

The NEPHELE DCN, shown in Fig. 1, is divided into P pods of racks and is built out of hybrid electrical/optical top-of-rack (TOR) switches and all-optical POD switches. A pod consists of I POD switches and W TOR switches, interconnected as follows: each TOR switch has I ports with tunable transmitters (Tx) and each one is connected to a different POD switch (among I). A rack consists of S servers connected through S corresponding ports to the TOR switch. The POD switches are interconnected through WDM rings to form optical planes. An optical plane consists of a single POD switch per pod (for a total of P POD switches) connected with R fiber rings. Each fiber ring carries WDM traffic of W wavelengths (by design equals the number of racks/pod). There are I identical and independent optical planes. In total, there are $I \cdot P$ POD switches, $W \cdot P$ TOR switches and $I \cdot R$ fiber rings.

The key routing concept is that each TOR switch listens to a specific wavelength and wavelengths are re-used among pods. Each TOR employs Virtual Output Queues (VOQ) per TOR destination to avoid head-of-line blocking. The NEPHELE TORs employ tunable Tx that are tuned according to the desired destination. Thus, the tunable Tx selects the destination/ wavelength to route traffic. The 1×2 space switch inside the corresponding POD switch is set according to whether the destination is in the same pod with the source or not. Intra-pod traffic is forwarded to an AWG that passively routes it towards the selected destination. Inter-pod traffic is routed via the 1×2 switch towards a $W \times R$ CAWG and then to one of the R fiber rings (according to the input port/source and the wavelength used). The traffic propagates in the ring passing through intermediate POD switches and is dropped at the destination pod, by setting appropriately the related WSSs. The drop port is connected to an AWG that again passively routes the traffic. Finally, NEPHELE operates in a *slotted and synchronous manner* as discussed in the next section. A parallel EPS network can also be utilized to handle high priority traffic and/or ACK TCP packets which, according to simulations presented in this paper, seem to play a major role in the network performance. A more detailed description of the NEPHELE data plane is provided in [17][18]. Also [17] presents some basic techno-economic results.

III. NEPHELE Control Cycle

NEPHELE exploits the SDN concept that decouples the data and control planes through open interfaces, enabling programmability of the networking infrastructure [17]. A key functionality of NEPHELE SDN controller is the coordination of the resource usage, including the timeslot/plane dimension [19]. Thus, an important building block of the SDN controller is the *scheduling engine*, which allocates resources for TOR communication in a periodic and on demand manner.

Two scheduling approaches are envisioned in NEPHELE.

We assume that long and medium term traffic variations can be solved with offline scheduling algorithms. The offline scheduling algorithms that run periodically or on demand (triggered by significant application/traffic changes) can calculate the optimum resource allocation (schedule) since their running time is not crucial. Then sort-time traffic variations or failure events are treated by faster online scheduling algorithms that calculate incremental changes in the running schedule.

Moreover, we also envision two traffic identification modes: (i) application-aware and (ii) feedback-based. The former mode [5], [6] assumes that applications communicate to the NEPHELE SDN controller (or via the DC orchestrator) their traffic requirements. The latter, feedback-based, mode assumes that the central controller collects (monitors) data from the TOR queues [9]. Hybrid versions of these two modes are also applicable.

In the NEPHELE network we divide the time in slots and we have periods of T timeslots. In all scheduling and traffic identification cases, we assume that the controller creates the *queue matrix* $\mathbf{Q}(n)$ (of size $(W \cdot P) \times (W \cdot P)$) for period n. We denote by $\mathbf{A}(n)$ the matrix of arrivals at the queues during period n, and by $\mathbf{S}(n)$ the schedule calculated for period n.

The NEPHELE network operates in *two parallel cycles*: a) Data communication cycles of T timeslots (also referred to as a *Data period*), where actual communication between TORs takes place and b) Control plane cycles of duration C (measured in Data periods), where control information is exchanged. Control plane cycle n corresponds to Data period n, and computes the schedule $\mathbf{S}(n)$ to be used during that period. Note, however, that the schedule is computed based on information that was available C periods earlier than the Data period the control plane cycle is applied to. Thus, $\mathbf{S}(n)$ is a function of $\mathbf{Q}(n-C)$, i.e.,

$$\mathbf{S}(n) = f\,(g[\mathbf{Q}(n-C)]), \qquad (1)$$

where $\widehat{\mathbf{Q}}(n) = g[\mathbf{Q}(n-C)]$ is the function that creates the *estimated queue matrix* $\widehat{\mathbf{Q}}(n)$ from $\mathbf{Q}(n-C)$, upon which the schedule is calculated, and f is the scheduling algorithm. When $C > 1$ period (control delay is larger than the Data period), a new Control plane cycle still starts every Data period. So, there are C Control plane cycles running in parallel.

The queues evolution is described by $\mathbf{Q}(n+1) = \mathbf{Q}(n) + \mathbf{A}(n) - \mathbf{S}(n)$, where $\mathbf{S}(n)$ is calculated as in Eq. (1). The control plane delay C depends on many factors, including the execution time of the scheduling algorithm, the delay of the control protocol carrying information from TORs to the SDN controller and from the SDN controller to the data plane devices. Both delays depend on the network size and the choice of the Data period T. For the scheduling decisions to be efficient, the scheduling matrix $\mathbf{S}(n)$ computed based on an estimated queue matrix $\widehat{\mathbf{Q}}(n)$ [which in turn is calculated from $\mathbf{Q}(n-C)$] should be a *"good"* scheduling to be used during Data interval n. This is true when $\widehat{\mathbf{Q}}(n)$ is a good approximation of $\mathbf{Q}(n)$. In case of slowly or moderately changing traffic, we expect calculations made for previous periods to be valid.

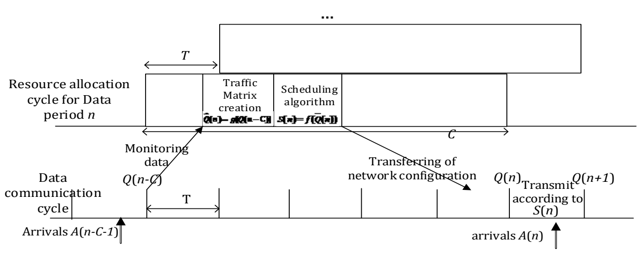
...
T
Resource allocation cycle for Data period n
Traffic Matrix creation
$\widehat{Q}(n)=g[Q(n-C)]$
Scheduling algorithm
$S(n)=f[\widehat{Q}(n)]$
C
Monitoring data
$Q(n\text{-}C)$
Transferring of network configuration
Data communication cycle
T
$Q(n)$
Transmit according to $S(n)$
$Q(n\text{+}1)$
Arrivals $A(n\text{-}C\text{-}1)$
arrivals $A(n)$

Fig. 2. NEPHELE Data and Control plane cycles.

In estimating $\widehat{\mathbf{Q}}(n)$ from $\mathbf{Q}(n-C)$, it is possible to use statistical predictions, filters, and other (notably, predefined cluster application communication patterns) methods to improve performance. Moreover, it is possible for the scheduler to *void fill* the unallocated resources in $\mathbf{S}(n)$ to enable opportunistic transmissions. Finally, the overall scheme is "self-correcting": if some queues are not served for some periods due to poor scheduling and their size grows due to new arrivals, this will be communicated with some delay to the controller, and they will eventually be served.

IV. Scheduling Algorithms

We now focus on the scheduling problem in the NEPHELE network. In any traffic identification mode (application-aware or feedback-based), we start from the estimated queue matrix $\widehat{\mathbf{Q}}(n)$ and devise algorithms to calculate the schedule $\mathbf{S}(n)$ (function f in Eq. (1)). For reference, we can assume that we calculate the estimated queue matrix (function g in Eq. (1)) as $\widehat{\mathbf{Q}}(n) = \mathbf{A}(n-C-1) + \widehat{\mathbf{Q}}(n-1) - \mathbf{S}(n-1)$, where in the expression we acknowledge that due to control plane delay C, the central scheduler has access to (delayed) arrival information $\mathbf{A}(n-C-1)$ instead of $\mathbf{A}(n)$. This corresponds to the case where the schedule $\mathbf{S}(n)$ calculated on $\widehat{\mathbf{Q}}(n)$ serves the arrived traffic $\mathbf{A}(n-C-1)$, plus a correction equal to traffic not served in the previous period $\widehat{\mathbf{Q}}(n-1) - \mathbf{S}(n-1)$.

The scheduling algorithm provides the schedule $\mathbf{S}(n)$, which identifies the TOR pairs that communicate *during each timeslot and for each optical plane* for Data period n. Note that wavelengths and rings are dependent resources; the selected wavelength is determined by the destination and the ring depends on the source and destination [18]. So, in NEPHELE, the allocated resources are the timeslots and the optical planes ($I \cdot T$ in total), also called *generalized slots*. The scheduling algorithm takes the estimated queue matrix $\widehat{\mathbf{Q}}(n)$ and decomposes it (fully or, if not possible, partially) into a sum of $I \cdot T$ *permutation matrices.* These identify the source and destination TORs that communicate at each generalized slot. The scheduling algorithm takes into account the constraints under which a TOR can transmit/receive to/from a single TOR.

As discussed earlier, there are two scheduling approaches: offline and incremental, trading off execution time for optimality.

A. Offline Scheduling

Offline scheduling pertains to the optimal decomposition of matrix $\widehat{\mathbf{Q}}(n)$. We define the *critical sum* $H[\widehat{\mathbf{Q}}(n)] = h$ as the maximum of the row sums and column sums of matrix $\widehat{\mathbf{Q}}(n)$. The decomposition of $\widehat{\mathbf{Q}}(n)$ can be performed in an optimal manner following the well-known Hall's theorem (an integer version of the Birkhoff-Von Neumann theorem). A more detailed analysis of these techniques is discussed in [18].

The column sums will be *on the average* $\leq S \cdot T$, if the destinations of packets are uniformly distributed, or with *high probability*, if the network operates at less than full load. Also, a flow control mechanism can be applied to smoothen the traffic going to a given destination and enforce this constraint. In such a ("typical") case, the column sums of the arrival matrix $\mathbf{A}(n)$ will be $\leq S \cdot T$ and so will also be its critical sum, and thus the schedule $\mathbf{S}(n)$, that is calculated based on $\widehat{\mathbf{Q}}(n) = \mathbf{A}(n - C)$, assuming $S \leq I$, can be chosen so as to completely serve all the arrivals in $\mathbf{A}(n - C)$ in the available $I \cdot T$ generalized slots. Thus, in the typical case, NEPHELLE provides both *full throughput and delay guarantees*.

In the worst case, the optimal algorithm executes a maximum matching algorithm $I \cdot T$ times. Finding a maximum matching with e.g. the well-known Hopcroft–Karp algorithm exhibits complexity of $O(M(\widehat{\mathbf{Q}}) \cdot \sqrt{W \cdot P})$, where $M(\widehat{\mathbf{Q}})$ is the number of nonzero elements in $\widehat{\mathbf{Q}}$, which in the worst case equals $(W \cdot P)^2$.

B. Incremental Scheduling

It is evident from the above discussion and related results [18] that offline scheduling is not suitable to serve short-term varying traffic. Measurements in commercial data centers indicate that application traffic can be relatively bursty, with flows activating/ deactivating within ms [4]. However, the traffic tends to be highly locally persistent: a server tends to communicate with a set of destinations that are located in the same rack or the same cluster/ pod [4]. Note that TOR switches in NEPHELE aggregate the flows of the servers in a rack, smoothening out the burstiness of individual flows, especially considering locality persistent traffic.

A detailed definition of locality persistency is given in [18]. We denote by $\mathbf{D}(n) = \mathbf{A}(n) - \mathbf{A}(n-1)$ the arrival matrix difference, and by $\delta(\cdot)$ the density of a matrix. Then, the *Locality Persistency Property* holds if $\delta(|\mathbf{D}(n)|) \ll 1$. We also define the estimated queue matrix difference as $\mathbf{D}_{\widehat{\mathbf{Q}}}(n) = \widehat{\mathbf{Q}}(n) - \widehat{\mathbf{Q}}(n-1)$. Note that when arrivals have the locality persistency property, then we also expect $\delta(|\mathbf{D}_{\widehat{\mathbf{Q}}}(n))|) \ll 1$.

Motivated from the high locality observation, we investigated incremental scheduling, i.e. rely on the previous schedule to calculate the new one. The expected benefit is that we need to update only changed elements of the permutation matrices of the decomposition of $\widehat{\mathbf{Q}}(n+1)$, with no need to modify the rest. A number of incremental scheduling algorithms are presented in [18] where we also present a greedy incremental heuristic with complexity of $O(\delta(|\mathbf{D}_{\widehat{\mathbf{Q}}}|) \cdot I \cdot T \cdot (W \cdot P)^2)$, where $\delta(|\mathbf{D}_{\widehat{\mathbf{Q}}}|) \ll 1$ in view of the persistency property.

This heuristic achieves throughput that is close to optimal and running time in the order of hundreds of ms [18], using Matlab and an Intel® Core™ i5 laptop. A parallel implementation of the heuristic algorithm on an FPGA was presented in [20] and showed that the schedule can be computed in tens of ms even for dense input matrices $(\delta(|\mathbf{D}_{\widehat{\mathbf{Q}}}| < 0.25)$ using incremental algorithms. This implies that we can calculate the schedule within 1 Data period, which is quite promising for the performance of the NEPHELE architecture. However, the control plane overhead C depends also on the signaling overhead: monitoring (in feedback based traffic estimation mode) and transferring the schedule to the data plane devices, the NEPHELE POD and TOR switches. The effect of the total control plane overhead is examined in the next section.

V. Performance evaluation

A. Simulation Model and parameters

To evaluate the performance of the NEPHELE architecture, we developed a packet level network simulator. The simulator is an extension of OMNET++ 4.3.1 with INET 2.4.0, a framework that contains implementations for various real-life network components and protocols. We evaluated the network performance using an application that simulates MapReduce, which was implemented by Mellanox.

In our simulation model, we consider that the control plane delay, which includes the time to gather monitoring information (if we operate the network in feedback based, would be zero in application-aware mode), to calculate the schedule (which as previously discussed is fast, within 1 Data period [20]) and to distribute the schedule to the data plane devices, is described through the parameter C. This in turn defines the number of multiple identical (virtual) schedulers that work in parallel. We also assume that each parallel scheduler knows the C previous schedules (feasible, as the schedule is computed in 1 Data period).

In the simulated network we run a number of MapReduce jobs simultaneously. Each MapReduce job requires a number of worker nodes: *mappers*, *reducers* and *storage servers* and runs for a number of iterations. The communication pattern for each particular MapReduce job, regarding the server where each worker node resides, the size of the MapReduce data produced in each phase, the number of MapReduce iterations and the computational delay for *map* and *reduce* operations, are described using appropriate semantics in an input file. In the simulations the assignment of the worker nodes to the servers was random. This means that a server could host simultaneously multiple types of worker nodes for the same or different jobs.

The communication between the worker nodes is achieved via Ethernet packets over TCP/IP. We assumed full-duplex 10G Ethernet from a server to the corresponding NEPHELE TOR switch. For the TOR to TOR communication we rely on NEPHELE TDMA operation. The Ethernet packets are stored in

Virtual output queues (VOQ) and served in slots according to the computed schedules.

We study the impact of various parameters, such as the Control cycle delay C, the number of MapReduce jobs, or the cluster size $(P \cdot W)$, on the throughput, in terms of total makespan. The makespan is defined as the time it takes for all MapReduce jobs to finish. TABLE I. summarizes the NEPHELE network parameters, as well as the TCP-related parameters. Note that a target for the NEPHELE network would be to have 1600 racks with 20 servers each, while each timeslot (of duration 200μs) aggregates the traffic of all servers residing in a rack. Since it is not possible to simulate a fully-fledged NEPHELE network, but only smaller clusters with fewer servers per rack, the NEPHELE parameters are also scaled down accordingly. We assumed $I = 2$ optical planes, and the scheduling period T took values so that the generalized slots/resources equals to the number of racks $(T \cdot I = P \cdot W)$.

The key parameters that we examine are the Control cycle delay C, the number of MapReduce jobs that run simultaneously in the cluster and the number of cluster's racks; their default values are 4, 5 and 8, respectively. In all scenarios, the ratios of the MapReduce worker nodes types remained the same: the number of mappers equals to half, while the number of reducers and storage servers equals to a quarter of the available servers. A parallel (dual) network (utilizing 1 Gbps capacity) is also used to route the TCP ACKs.

TABLE I. SIMULATION PARAMETERS

Parameter			Value		
Number of servers in each rack (S)			2		
Number of planes (I)			2		
Link capacity per plane (each direction)			10Gbps		
Timeslot duration			200μs		
Maximum segment size (MSS)			625 bytes		
TCP window size			65000 bytes		
Storage server	Mapper	Reducer output	5	10	5 Mbytes
Mapper processing time			25μs		
Reducer processing time			20μs		
Number of MapReduce iterations			3		

We examine three queue matrix estimation policies. The first estimation policy assumes static uniform traffic under which no traffic identification mode (monitoring or application awareness) is assumed and the resource allocation is evenly distributed among the TOR pairs (round-robin scheduling). The second policy assumes that $\widehat{\mathbf{Q}}(n)$ (described in Sections III and IV) is computed based on the most recent known arrivals $\mathbf{A}(n - c - 1)$. The third policy is a simplistic prediction mechanism that assumes that the arrivals for the next C Data periods will be equal to the latest $\mathbf{A}(n)$. It then virtually applies the latest C known schedules and computes an estimation for the remainder in the queues when the schedule will be applied (after C Data periods). The above queue estimation policies are combined with the incremental scheduling algorithm which is extended with a greedy randomized void filling heuristic. Void filling is used to fill the unallocated slots left empty by the scheduling algorithm. In particular, a randomized greedy heuristic greedily computes a set of matchings in order to fill the free slots in an uniform way, taking into account the previously allocated slots and the transmission constraints that they yield.

B. Simulation Results

We initially examine the effect of utilizing i) a parallel packet switched network over which we sent TCP ACK packets and ii) a randomized void filling heuristic to fill the empty slots/ permutations of the schedules on slot (network capacity) utilization over time. As it can be observed in Fig. 3, both the effect of the parallel network and the randomized void filling heuristic is quite significant. Since, TCP features congestion control, the TCP window limits the traffic load the servers transmit. This has a major impact to the overall slot utilization and thus to the throughput and the makespan of the network. These two techniques improve the TCP window pipelining resulting to improved slot utilization and reduced makespan. In particular, we observed a reduction of the makespan for the 4 MapReduce jobs from ~27,4 s in the case of *no parallel/no void filling* to ~27,2 s in the case of *parallel/no void filling* and to ~14 s in the case of *no parallel/void filling*. The combination of *parallel/ void filling* achieves a substantially lower makespan of ~10,3 s. In the following we will assume that the NEPHELE network uses both *parallel/ void filling*.

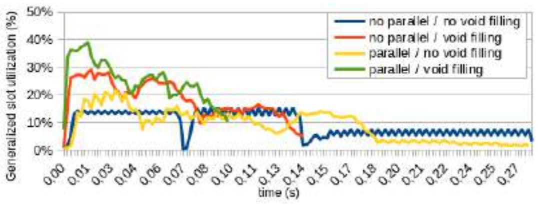

Fig. 3. Impact of the parallel network and randomized void filling heuristic on slot utilization.

We now examine the effect of the control delay C which was varied from 0, 5, 10, 20, 50 to 200 Data periods. As it is shown in Fig. 4, the makespan for the case of the static round-robin policy remains constant at about 0.36 s, regardless of the Control cycle delay. Meanwhile, the other two policies seem to perform better for at most 19%, given that they take into account the traffic (monitoring or application awareness) and carry out scheduling based on $\widehat{\mathbf{Q}}(n)$ estimates. This performance improvement decreases as the Control cycle delay increases, and eventually in the sample of Control cycles equal to 200 Data periods, it gets worse than the static round-robin for at most 13%. This is expected, since the longer control delay results to an increased chance the actual traffic at the queues to substantially differ from the calculated schedule. It can also be observed that in small numbers of Control cycles, utilizing prediction also improves the performance. However, this improvement fades out from 20 Control cycles and on.

In the next scenario, we consider the cases where we have 1, 4, 7 and 10 MapReduce jobs simultaneously running on the cluster. It is expected that as the number of jobs increases, the network load increases, but also the traffic dynamicity decreases, given that the assignment of the worker nodes with the servers is done randomly and uniformly. As shown in Fig. 5, the makespan increases with the job number in all queue matrix estimation policies, since the network load increases. However, especially in the case of 1 job, where only certain parts of the network are utilized in each Mapreduce phase, we can see that the static round-robin policy performs much worse than the other two policies for about 32%. This difference is reduced for larger numbers of jobs to at least 16%.

In the last considered scenario, we have different cluster sizes, namely of 4, 8, 16 and 32 racks (8, 16, 32 and 64 servers, respectively). Fig. 6 shows the performance of the three queue matrix estimation policies. In particular, we can observe that the policies that take into account the traffic have a much better performance than the static round-robin that ranges between 12-48% and increases with the increase of the cluster size.

In our tests we compared NEPHELE with a fat-tree topology. We observed that when Control cycle was set to 0, NEPHELE performed similarly to a fat-tree, in terms of makespan.

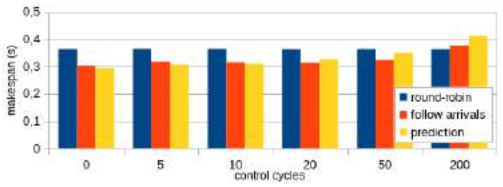

Fig. 4. Impact of control cycle (in Data periods) on makespan.

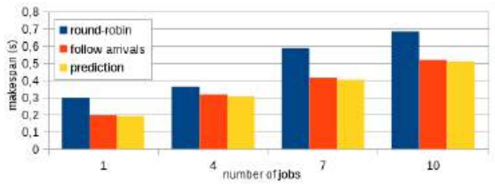

Fig. 5. Impact of the number of MapReduce jobs on makespan.

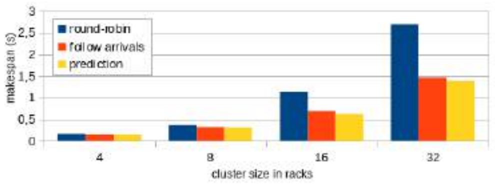

Fig. 6. Impact of cluster size on makespan.

VI. Conclusions

We presented the NEPHELE DCN architecture and described the related resource allocation problem. In NEPHELE, a centralized SDN controller allocates slots and optical planes to communicating pairs, and thus coordinates over time, space and wavelength to avoid collisions and achieve efficient operation. We described the NEPHELE control cycle, including the importance of the policy used to obtain good queue matrix estimates that approximate the traffic pattern after the control cycle delay. We conducted simulations using OMNET++ under MapReduce realistic traffic. We examined the effect of utilizing a parallel network for TCP ACKs, and of a void filling heuristic. We observed that both these techniques, improve the makespan. We considered the case of applying a static round-robin policy and two policies that take into account the traffic. We observed that when the control cycle delay is high, a static round-robin policy seems preferable. The policies that take into account the traffic induce a significant improvement to the total makespan that can reach 48% when the short-term load dynamicity is high.

Acknowledgment

Partially funded by EC through NEPHELE project (g.a. 645212).

References

[1] Cisco Global Cloud Index: Forecast and Methodology, 2016-2021.

[2] Al-Fares, A. Loukissas, A. Vahdat, “A Scalable, Commodity Data Center Network Architecture”, ACM SIGCOMM, 2008

[3] T. Benson, A. Akella, D. Maltz, “Network Traffic Characteristics of Data Centers in the Wild”, ACM SIGCOMM, 2010.

[4] A. Roy, H. Zeng, J. Bagga, G. Porter, A. Snoeren, “Inside the Social Network's (Datacenter) Network”, ACM SIGCOMM, 2015.

[5] J. Follows, D. Straeten, “Application driven networking: Concepts and architecture for policy-based systems”, IBM Corporation, 1999.

[6] X. Zheng, Z. Cai, J. Li, H. Gao, “An application-aware scheduling policy for real-time traffic”, International Conference on Distributed Computing Systems (ICDCS), 2015.

[7] C. Kachris, I. Tomkos, “A Survey on Optical Interconnects for Data Centers”, IEEE Communications Surveys & Tutorials, 14 (4), 2012.

[8] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, A. Vahdat, “Helios: a hybrid electrical/optical switch architecture for modular data centers”. ACM SIGCOMM, 2010.

[9] G. Wang et al, “c-through: part-time optics in data centers”, ACM SIGCOMM, 2010.

[10] K. Christodoulopoulos, D. Lugones, K. Katrinis, M. Ruffini, D. O’Mahony, “Performance Evaluation of a Hybrid Optical/Electrical Interconnect”, IEEE/OSA Journal of Lightwave Technology, 2015.

[11] Y. Ben-Itzhak, C. Caba, L. Schour, S. Vargaftik, “C-Share: Optical Circuits Sharing for Software-Defined Data-Centers”, arXiv, 2016.

[12] N. McKeown et al, “OpenFlow: Enabling Innovation in Campus Networks”, ACM Computer Communication Review, 2008.

[13] A. Singla et al, “Proteus: a topology malleable data center network,”, ACM SIGCOMM Workshop on Hot Topics in Networks, 2010.

[14] S. Peng, et al, “Multi-Tenant Software-Defined Hybrid Optical Switched Data Centre”, IEEE/OSA Journal of Lightwave Technology, 2015.

[15] N. Calabretta, W. Miao, “Optical Switching in Data Centers: Architectures Based on Optical Packet/Burst Switching”, Optical Switching in Next Generation Data Centers, pp.45-69, Springer, 2017.

[16] G. Porter et al, “Integrating microsecond circuit switching into the data center,” ACM SIGCOMM, 2013.

[17] P. Bakopoulos, et. al. “NEPHELE: an end-to-end scalable and dynamically reconfigurable optical architecture for application-aware SDN cloud datacenters”, IEEE Communications Magazine, 2018.

[18] K. Christodoulopoulos et al, "Efficient bandwidth allocation in the NEPHELE optical/electrical datacenter interconnect", IEEE/OSA Journal of Optical Communications and Networking, 9(12), pp. 1145-1160, 2017.

[19] G. Landi, M. Capitani, K. Christodoulopoulos, D. Gallico, M. Biancani, M. Aziz, “An Application-Aware SDN Controller for Hybrid Optical-Electrical DC Networks”, ICN 2017.

[20] I. Patronas, V. Kitsakis, N. Gkatzios, D. Reisis, K. Christodoulopoulos, E. Varvarigos, "Scheduler Accelerator for TDMA Data Centers", PDP 2018.