Evaluation of Datacenter Network Topology Influence on Hadoop MapReduce Performance

Zdenek Kouba, Ondrej Tomanek, Lukas Kencl
Dept. of Telecommunications Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Email: {zdenek.kouba, ondrej.tomanek, lukas.kencl} [at] fel.cvut.cz

***Abstract*—Hadoop MapReduce has nowadays become the de-facto standard for the Big-Data processing within Cloud datacenters. However, little is known about the influence of datacenter network topology on Hadoop performance, and suitability of various topologies for different workload distributions. By extending a publicly available simulator CloudSim, we simulate six well-known or recently proposed topologies (Hierarchical, FatTree, DCell, CamCube, BCube, MapReduce) and evaluate Hadoop MapReduce performance across varyingly distributed workloads. We conclude that while no topology is clearly optimal, the experimental CamCube topology exhibits the most promising results. However, different topologies correspond to different workload divisions in terms of best performance, with greatly differing results, and generally weak performance under highly skewed workloads. This finding could lead to significant Hadoop MapReduce performance improvements by adjusting or selecting appropriate datacenter network topologies – potentially even at runtime, using Software Defined Networking.**

I. Introduction

A lot of Internet computation and communication has already moved into datacenters and this trend of co-located workloads is set to continue. MapReduce [1] is among popular data and communication-intensive applications, but it has a large and inter-related configuration design space. This means that the relationship between servers, available bandwidth and MapReduce performance is not straightforward. One of the factors which MapReduce is sensitive to is the network topology that underlies its allocated resources and backgrounds the traffic from co-located workloads. It has been shown that by exposing topology information to MapReduce systems, VM locality and average execution times improve, indirectly translating to monetary savings in executing job on cluster. But there is still a gap in understanding the interplay between MapReduce and specific datacenter network topologies. We search the set of standard and experimental popular topologies for the best-performer in terms of the job finish time. We also provide scalability comparison and distances from job finish times on the baseline topology.

MapReduce jobs are comprised of several phases, of which three involve a network data transfer (as depicted in Fig. 1). Moreover, the data transfer in the second phase (commonly referred as Shuffle/Merge) is generally order of magnitude higher than in the other two and involves a distributed serialization barrier blocking the subsequent phase's execution. As such, MapReduce jobs are prone to creating severe bottlenecks unless the network is provisioned properly. Also, there is no one-size-fits-all approach, as there are a number of job-specific variables such as data volume, workload type or Reducer–workload division. Fine-tuning MapReduce performance is thus a multi-objective optimization problem and studying the design space using actual clusters is challenging given equipment costs, extensive configurations, setup times and need to redo everything for different cluster. In this work we conduct series of simulations to study the dependence of performance of distributed computing tasks following the MapReduce model on Reducer–workload division, job dimension and network topology.

We use *Hadoop MapReduce* [2] as it basically serves as the industry-default implementation. We use *CloudSim* [3] simulator for running our simulations. Our work builds on and extends the work described in thesis [4]. The contributions of this work are:

- Simulation architecture for simulating topology influence on Hadoop MapReduce performance;
- Open-source implementation of the *CloudDynTop* simulator module;
- Comparison of six popular topologies under varying Reducer–workload divisions and varying datacenter network sizes.

The remainder of the paper is organized as follows: § II provides an overview of the related work. § III describes MapReduce data flows and considered topologies. § IV describes important aspects of the simulation architecture, § V shows the simulation setup and § VI details the results. We conclude and discuss implications in § VII.

II. Related work

Many system-design aspects drive MapReduce performance. Among important ones are task scheduling, skew mitigation, Shuffle/Merge optimization and choice of topology.

Task scheduling should leverage data locality in face of host failures, in order to avoid moving large volumes of data around. *Delay scheduling* [5], *Bandwidth-aware scheduling* [6] and VM placement algorithms [7] represent approaches to achieving this. Skew corresponds to an uneven distribution of input data and/or computational load, resulting in hosts with tasks taking extraordinarily long to complete (stragglers). *SkewTune* algorithm [8] detects stragglers and redistributes the

unprocessed remainder of their input amongst new set of tasks. Shuffle/Merge is usually the most data-transfer-heavy phase of MapReduce processing. Priority queuing [9] or load balancing [10] can be used to the expedite this phase.

Datacenter network topology is an important design decision to make in MapReduce deployments. Aside from proven standard topologies, several promising experimental datacenter architectures and topologies were proposed previously. CamCube [11] builds on 3D Torus topology, allows services to supply their own routing algorithm and provides server-based routing and switching. DCell [12] is a recursively defined topology and BCube [13], another server-centric topology, allows for commodity network gear. Both the standard and the said experimental topologies were considered in our work.

A number of MapReduce simulations with varying optimization objectives and topologies considered were conducted previously. DCell, FatTree and Hierarchical topologies were compared in terms of the network delay and throughput [14]. *MRPerf* MapReduce simulator was used to study the influence of Star, Double rack, Hierarchical and DCell topologies on MapReduce performance [15]. MapReduce configuration design space was used to optimize MapReduce performance [16]. Topology-aware architectures [17], [18] were introduced to forecast performance of and to optimize MapReduce resource allocation.

In this work, we search for the best performing real–world topology in terms of the job finish time under various Reducer–workload divisions and network sizes.

Several MapReduce simulators exist, differing in network simulation engine used, programming language, scalability, level of detail and skew support. *MRPerf* [15] offers good scalability, *MRSim* [19] a good level of detail and *MRSG* [20] great scalability and skew support. *CloudSim* [3] offers skew support together with great scalability and as such was chosen for the purpose of our work.

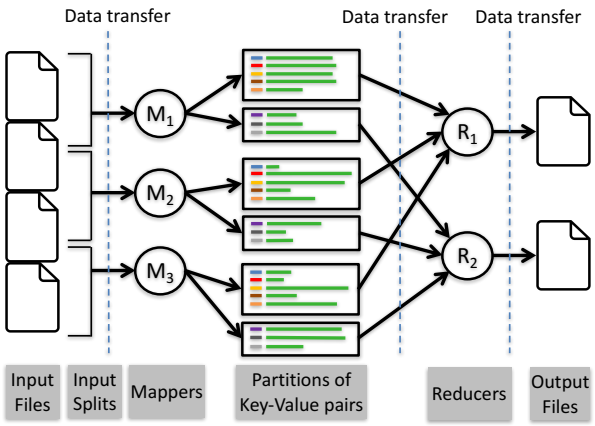


Fig. 1. *MapReduce job data flow*. Job input is distributed among Mappers, who then produce an intermediate workload distributed among Reducers, who in turn produce the output. These three phases generally involve a network data transfer, with the second phase (Shuffle/Merge) being most data-transfer-heavy. The Mapper workload is typically uniform. The Reducer workload is generally non-uniform due to differing sizes of Mapper-produced partitions.

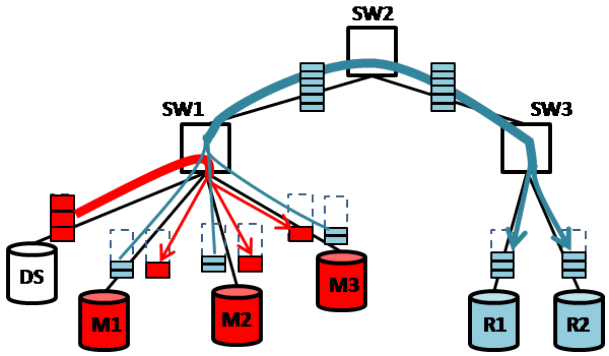


Fig. 2. *Poorly-performing, underprovisioned Hierarchical topology*. Resultant are network bottlenecks on paths from the Data Source DS to Mappers M_i (red) and from Mappers M_i to Reducers R_r (blue). The amount of unused bandwidth is illustrated via incomplete rectangles. All links are of an equal capacity and switches forward data at wire speed.

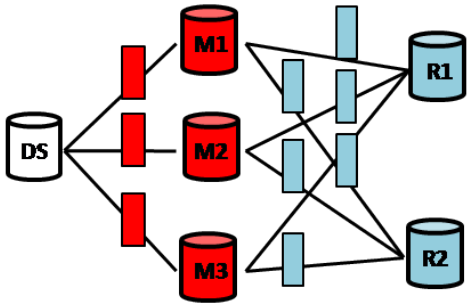


Fig. 3. *Well-performing, well-provisioned topology*. Each flow has its own dedicated link, therefore there are no bottlenecks, the network is fully utilized and job completes in a shortest time possible. In fact this is the hypothetically optimal topology for MapReduce job processing.

III. Data flow and Topologies

MapReduce [1] is a programming model for distributed processing of large data volumes. Many real world tasks are expressible in this model. A MapReduce job consists of M Map tasks and R Reduce tasks. Their hosts are called Mappers and Reducers, respectively. Each Mapper is assigned a portion of the job's input (typically distributed uniformly). The data are parsed as key-value pairs and Mappers process them one pair at a time. Mappers produce on their output a new set of key-value pairs, organized into R partitions (generally of an unequal size) according to the following formula:

$$r = hash(key) \bmod R$$

Following is the Shuffle/Merge phase, where the created partitions (comprising the so-called *Reducer workload*) are fetched by the respective Reducers (rth partitions by the rth Reducer) and therein sorted by keys. A Reducer can start processing its input only after all Mappers have finished (end of Map phase) and its input has been sorted (end of Shuffle/Merge phase). Similarly to Mappers, Reducers process their input one key-value pair at a time and produce a new set of key-value pairs, which represents the final output of the MapReduce job. Fig. 1 illustrates the entire process with $M = 3$ and $R = 2$ tasks. Figures 2 and 3 represent two different clusters, yielding a very different job finish times because of the different underlying network topology.

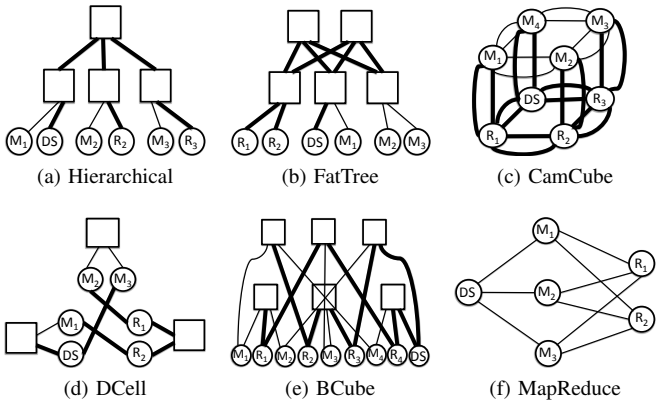


Fig. 4. *Illustration of topologies considered in the simulations.* Circles denote hosts, squares denote switches and lines denote links of two capacities (distinguished by line weight).

We thoroughly evaluate the influence which the popular datacenter network topologies have on Hadoop MapReduce performance. Considered are two standard, three experimental and a single hypothetical topology (illustrated in Fig. 4):

- *Hierarchical*
- *FatTree*
- *CamCube*
- *DCell*
- *BCube*
- *MapReduce*

Hierarchical and FatTree topologies are variations on a standard n-tier Tree. Another Tree-looking topology is a recursive BCube [13], defined by parameters N (number of hosts in the level-0 BCube) and K (index of the highest recursion level). $N = 3, K = 1$ BCube is depicted in Fig. 4. CamCube [11] is based on 3D Torus topology, where servers are directly interconnected. DCell [12] topology combines switches and server routing. The hypothetically optimal, "MapReduce", topology is modeled after the MapReduce data flow, providing a dedicated connection between Data Source and Mappers as well as between Mappers and Reducers such that no link is traversed by more than a single flow.

IV. Simulator architecture

An extension of the *CloudSim* simulator, *CloudSimEx MapReduce*, was chosen for running our simulations. Despite some notable features, it lacked a sufficient level of datacenter detail and had a number of other deficiencies (scheduling, throughput allocation, missing topologies and static link capacities), all of which had to be addressed prior to our experiments. We thus implemented an open-source extension module *CloudDynTop*. The remainder of this section describes the important aspects of its simulation architecture (see [4] for the code as well as all four algorithms discussed).

A. Throughput allocation

CloudSimEx MapReduce uses Fair share allocation for assigning link's throughput to passing flows. Allocating only a fair share of a link of the lowest capacity among links the flow passes through would lead to sub-optimal utilization of some links in the network. In *CloudDynTop* we thus implemented *Available share allocation* that allows for assigning more than a fair share of link's capacity to a flow if possible, yielding shorter job finish times. The difference between the Fair share and the Available share allocation is illustrated in Fig. 5.

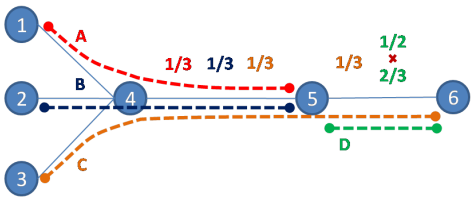


Fig. 5. *Available share versus Fair share throughput allocation.* All links are of a same capacity. According to the Fair share allocation, throughput of a flow C is determined by the link between hosts 4 and 5, therefore flow C can only use $1/3$ of capacity of the link between hosts 5 and 6. Available share allocation will assign the remaining $2/3$ to flow D, whereas Fair share allocation would only assign $1/2$ of the link's capacity to flow D.

B. Job Finish Time calculation

CloudDynTop keeps track of the simulated topology and, based on input in a form of scheduling (mapping between tasks and VMs) and provisioning plan (mapping between VMs and hosts), calculates the time needed for the tasks included in a scheduling plan to finish. The scheduled tasks are used to construct Flow and Computation instances. A Flow represents data transmission between two hosts and Computation represents data processing on a single host. Each Flow and Computation keeps track of its respective time to finish. Computation derives it from the number of remaining instructions and MIPS of the host running the VM hosting the task. Flow derives it from the number of MBs yet to be transmitted and from a bottleneck throughput and delay of the path taken through the network. The bottleneck throughput of a path is determined via Available share allocation algorithm (§ IV-A). Each Flow and Computation also has a list of prerequisite Flows and Computations. The running time of such set of processes is then computed using the Algorithm 1.

C. Routing

The Floyd-Warshall algorithm [21] determines which links a flow between two hosts will traverse. This algorithm is only ran once, before the MapReduce job. All the routes are thus precalculated. The link cost is calculated using the formula following the EIGRP routing protocol [22] link cost calculation.

D. Scheduling

CloudSimEx MapReduce provides a Branch-and-Bound algorithm for selection of the optimal scheduling plan prior to running a MapReduce job. The provided implementation only tested a candidate scheduling plan's running time against a predefined threshold configured by user. We improved this algorithm so that the running time is evaluated at each node of the scheduling tree, i.e. for each candidate scheduling plan (even incomplete). The branch search only proceeds if the

Algorithm 1 Job finish time of a set of processes

Input: $proc$: collection of Flows and Computations

$queued \leftarrow proc$
$running \leftarrow$ empty list
while $queued$ ¬empty $\vee$ $running$ ¬empty **do**
  for p in $queued$ **do**
    if p.allPrerequisitesFinished **then**
      $p.state \leftarrow RUNNING$
      $running$.add(p)
      $queued$.remove(p)
      if p is Flow **then**
        for $link$ in $p.path$ **do**
          $link.passingPathsCnt$++
        end for
      end if
    end if
  end for
  sort $running$ by remaining time (ascending)
  $rt \leftarrow$ smallest remaining time among $running$
  for p in $running$ **do**
    p.updateProgress(rt)
    if p.hasFinished **then**
      $running$.remove(p)
      if p is Flow **then**
        for $link$ in $p.path$ **do**
          $link.passingPathsCnt$−−
        end for
      end if
    end if
  end for
end while

running time of that node is shorter than the shortest time found at leaf nodes so far.

For the sake of performance, the search space is traversed concurrently and two constraints are introduced: a maximum of one single task can be scheduled on any given host and Map task-placement permutations are eliminated (permuting Mappers does not change the duration of flows from Data Source to Mappers). In contrast, all permutations in Reducers placement are evaluated, which is sufficient for exploring all possible relative positions of Mappers and Reducers due to the uniform connection of hosts to the network (every host sees the same labeled graph - a property of the topologies we are considering). This constraint reduces the search space significantly, as the number of Reducers in MapReduce scenarios is almost always smaller than the number of Mappers.

V. Simulation setup

A. Topologies

Sizes of the simulated networks are equal to the job dimension, i.e. $M + R + 1$ (number of Mappers plus number of Reducers plus the Data Source). Simulated are the following cases: $(M; R)$ = $(10; 3)$, $(15; 4)$ and $(20; 5)$. Larger topologies were not considered, as our scheduler (§ IV-D) exhaustively

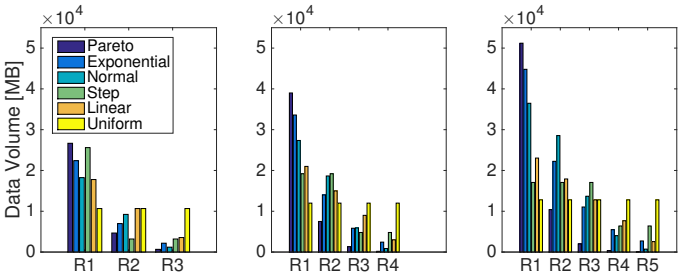


Fig. 6. *Example of the generated Reducer–workload divisions.* Each plot represents a single network size (i.e. job dimension). Within each plot, the volume of data a particular Reducer receives from every single Mapper is denoted by a single-colored bar. Considered workload divisions adhere to PDFs of six data distributions (distinguished by color).

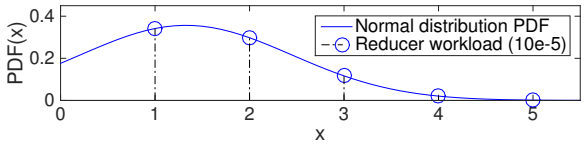


Fig. 7. *Normal Reducer–workload division generation.* For j Reducers, the corresponding PDF is evaluated at j pseudo-equidistant points on interval $(0; 5\rangle$, yielding the desired Reducer workload. This example shows $j = 5$.

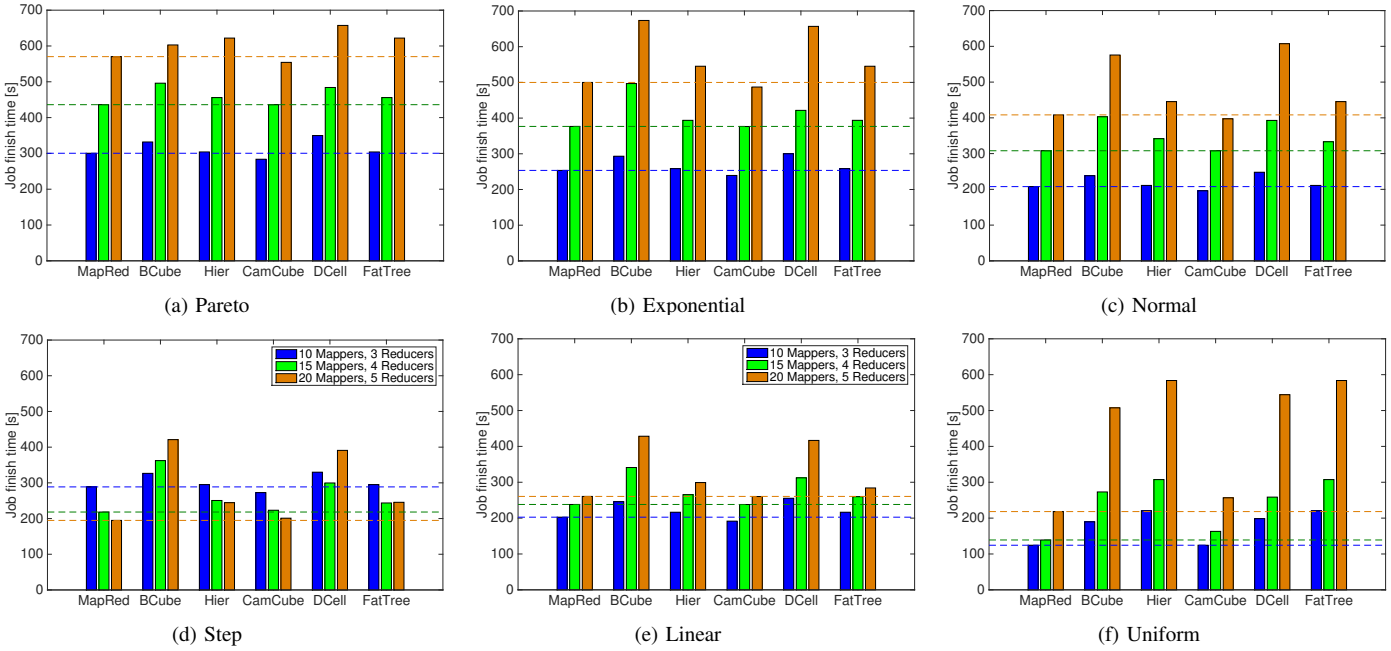
searches for the optimal scheduling plan and its running time thus grows significantly with a growing network size. Scheduler running time is not part of the job finish time.

BCube topology parameter values $(N; K)$, corresponding to the aforementioned network sizes, are $(4; 2)$, $(5; 2)$ and $(6; 2)$, respectively. Hierarchical and FatTree topologies have three tiers and a fan-out of four. To avoid introducing artificial bottlenecks, all the topologies except MapReduce are well-provisioned, i.e. Reducers (handle large data volumes), Data Source (generates large data volume) and switch interconnections (share oversubscribed links) are connected via 10Gbps links. The remainder of links are 1Gbps.

B. Workload

For the purpose of simulations we chose the Expansion workload [23], because it stresses the network and network bottlenecks become much more pronounced. Specifically, after the Data Source uniformly distributes the job input among Mappers, a total intermediate data volume (to be shipped across the network from Mappers to Reducers) is n times the size of the job input. We selected $n = 5$, yielding a large–enough realistic Reducer workload. Reducers keep their output locally and do not initiate any further network data transfer. To minimize the influence of the host computation on the job finish times, the computational load of this workload is small compared to the network load. Three job dimensions were simulated (§ V-A).

Examples of the generated Reducer–workload divisions are plotted in Fig. 6. The divisions adhere to PDFs of six data distributions - *Pareto, Exponential, Normal, Step, Linear, Uniform*. The distribution input parameters are set in a way that the sum of Reducer–input workloads equals the whole intermediate workload to be distributed. Pareto division splits the total workload according to the 80:20 rule between the first Reducer and the rest, reapplying this rule recursively. To generate Exponential and Normal divisions for j Reducers,



(a) Pareto (b) Exponential (c) Normal

(d) Step (e) Linear (f) Uniform

Fig. 8. *Job finish times.* Results of simulation runs of MapReduce job, varying three job dimensions (14, 20, 26 hosts), six Reducer–workload divisions (Pareto, Exponential, Normal, Step, Linear, Uniform) and six datacenter network topologies (MapReduce, BCube, Hierarchical, CamCube, DCell, FatTree).

the respective PDF is evaluated at j pseudo-equidistant points on interval $\langle 0; 5\rangle$ (Fig. 7 illustrates the Normal division). Step division splits Reducers into two similarly-sized groups within each the individual Reducers are loaded equally and the load ratio between the two groups is 4:1. Linear division loads Reducers in a linearly-increasing fashion, according to the $y = 2x + 1$ curve. As a result we get a representative spectrum of commonly occurring network workloads with different skew.

C. Constraints

The said simulation constraints, imposed to keep the topology a dominant contributor to the job finish times and to keep the scheduler running time short, can be summarized as:

- Pruning the scheduler search space
- A maximum of a single task slot per host
- A maximum of one task assigned to any single host
- Large Reducer workload
- Low computational load

VI. Evaluation

A set of 108 simulation scenarios was designed that varies three job dimensions, six Reducer–workload divisions and six network topologies. Only a single simulation run per scenario was conducted, since using the optimal scheduling plan (§ IV-D) and generating workload in a deterministic way (§ V-B) eliminate significant variations. This section presents results of the simulations, measured by *job finish time*.

Fig. 8 shows job finish times of job simulation runs. Three horizontal dashed-lines within every plot mark the job finish time on 1Gbps MapReduce topology, providing the baseline.

Shortest job finish times (~120 seconds) were achieved under Uniform Reducer–workload division. Linear and Normal divisions followed with the shortest job finish times starting at ~200 seconds. Highly-skewed divisions started at ~250 seconds (Exponential) and ~300 seconds (Pareto). Except the Uniform division, finish times of a job of the smallest dimension ($M = 10; R = 3$) were similar across all topologies.

Step division is somewhat special in that it is the only one, whose maximum workload per Reducer declines with a growing job dimension (i.e. network size), as shown in Fig. 6. As an implication, the job finish times were often slowly improving with the growing job dimension (MapReduce, CamCube) and plateauing at the largest simulated job dimension (Hierarchical, FatTree). In the case of DCell topology, the advantage of declining maximum workload appears to be dominated by the properties of the topology somewhere between ($M = 15; R = 4$) and ($M = 20; R = 5$). BCube topology does not seem to benefit from declining maximum workload as its job finish times kept increasing, likely due to oversubscribed links and congestion at switches. Overall, the Step division is essentially incomparable to other divisions.

CamCube was the only topology able to match the performance of 1Gbps MapReduce topology under most Reducer–workload divisions. The reason is the high number of links and high ratio of high-capacity links this 3D Torus has, embedding the MapReduce topology quite well. Performing similarly in all scenarios were the Hierarchical and FatTree topologies, because of their similar structure. The higher number of roots and core links did not prove too much of advantage for FatTree. This is because the high-capacity links at the core layer were able to handle the data volume.

All topologies displayed poor scalability under highly-skewed workload divisions (Pareto, Exponential). Respective variations in job finish times from the smallest to the largest network size were as high as 350 seconds. The skew itself is what causes such a limited scalability. Impacted most were DCell (consonant with observations in [14]) and BCube.

Under unskewed divisions except Normal division, CamCube exhibited best scalability. Hierarchical and FatTree displayed good scalability under Linear workload division.

All in all, the experimental topologies did not quite outperform the standard topologies. CamCube was overall the best performer and also displayed a good scalability. Hierarchical and FatTree topologies performed alike. Workload skew hampers scalability of all topologies (CamCube least) and increases variability of the job finish times. Scaling poorly under Uniform division and introducing job tails in general were topologies that are not highly-connected (BCube, DCell, Hierarchical, FatTree).

VII. Conclusion

The experimental results clearly show that the datacenter network topology can have major impact on Hadoop MapReduce job finish time, in some instances even doubling the time required. Out of all topologies considered, the experimental CamCube topology has performed best, owing to its high number of links. The frequently used FatTree topology has displayed reasonable performance, while the well-known DCell topology seemed to be rather outperformed by others.

Perhaps even more significantly, the results also document the fundamental influence of the distribution of workload among the Reducers on job finish time. Relative performance differences among topologies are greater under the less-skewed workloads. None of the topologies seems to handle skewed workloads particularly well, despite the fact that these are common in practice, and such workloads take much longer to finish than the more evenly distributed ones. Clearly, the static topology design has difficulties in coping with such high concentration of network traffic and computation on fewer links and hosts.

The presented study could definitely be improved, by trying alternative capacities of network links, or using tasks of greater size, or alternating the workload distributions more, or changing workload type, or analyzing logs and pinpointing bottlenecks. However, the results would probably not differ significantly, as they mostly reflect the fundamental properties of the studied topologies, and correspond to other published works [14], [18]. More simulation runs would not make a difference due to our workload generation strategy and scheduling according to the optimal scheduling plan. It is not clear how will the results scale with larger network sizes.

We hope this work can serve as inspiration for research on the future datacenter topology design. Perhaps Software Defined Networking can be employed to dynamically adjust the topology based on knowledge of the workload distribution. Or, different, highly adaptive resource provisioning mechanisms may be used for the cases of highly skewed workloads.

Acknowledgments

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS15/153/OHK3/2T/13. We thank anonymous reviewers for their comments.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, 2008.

[2] "Hadoop: Open source implementation of MapReduce," https://hadoop.apache.org/.

[3] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience, Wiley*, 2011.

[4] Z. Kouba, "Optimization of Network Architecture for Hadoop Performance," Master's thesis, Czech Technical University in Prague, 2015, http://hdl.handle.net/10467/61573.

[5] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling." ACM EuroSys, 2010.

[6] P. Qin, B. Dai, B. Huang, and G. Xu, "Bandwidth-Aware Scheduling with SDN in Hadoop: A New Trend for Big Data," *IEEE Systems Journal*, 2014.

[7] M. Alicherry and T. Lakshman, "Optimizing Data Access Latencies in Cloud Systems by Intelligent Virtual Machine Placement," in *IEEE INFOCOM*, 2013.

[8] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "SkewTune: Mitigating Skew in MapReduce Applications," in *ACM SIGMOD*, 2012.

[9] Y. Wang, X. Que, W. Yu, D. Goldenberg, and D. Sehgal, "Hadoop Acceleration Through Network Levitated Merge," in *IEEE/ACM SC*, 2011.

[10] Y. Fan, W. Wu, H. Cao, H. Zhu, W. Wei, and P. Zheng, "LBVP: A Load Balance Algorithm based on Virtual Partition in Hadoop Cluster," in *IEEE APCloudCC*, 2012.

[11] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic Routing in Future Data Centers," *ACM SIGCOMM*, 2011.

[12] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers," *ACM SIGCOMM*, 2008.

[13] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," *ACM SIGCOMM*, 2009.

[14] K. Bilal, S. U. Khan, L. Zhang, H. Li, K. Hayat, S. A. Madani, N. Min-Allah, L. Wang, D. Chen, M. Iqbal *et al.*, "Quantitative Comparisons of the State-of-the-art Data Center Architectures," *Concurrency and Computation: Practice and Experience, Wiley*, 2013.

[15] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "A Simulation Approach to Evaluating Design Decisions in MapReduce Setups," in *IEEE/ACM MASCOTS*, 2009.

[16] H. Herodotou and S. Babu, "Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs," *Proceedings of the VLDB Endowment*, 2011.

[17] G. Lee, N. Tolia, P. Ranganathan, and R. H. Katz, "Topology-Aware Resource Allocation for Data-Intensive Workloads," in *ACM APSys*, 2010.

[18] M. Li, D. Subhraveti, A. R. Butt, A. Khasymski, and P. Sarkar, "CAM: A Topology Aware Minimum Cost Flow Based Resource Manager for MapReduce Applications in the Cloud," in *ACM HPDC*, 2012.

[19] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu, "MRSim: A Discrete Event based MapReduce Simulator," in *IEEE FSKD*, 2010.

[20] W. Kolberg, P. D. B. Marcos, J. C. Anjos, A. K. Miyazaki, C. R. Geyer, and L. B. Arantes, "MRSG - A MapReduce Simulator over SimGrid," *Parallel Computing, Elsevier*, 2013.

[21] R. W. Floyd, "Algorithm 97: Shortest Path," *Communications of the ACM*, 1962.

[22] R. Albrightson, J. Garcia-Luna-Aceves, and J. Boyle, "EIGRP - A Fast Routing Protocol Based on Distance Vectors," *Interop 94*, 1994.

[23] K. Ren, G. Gibson, Y. Kwon, M. Balazinska, and B. Howe, "Hadoop's Adolescence: A Comparative Workloads Analysis from Three Research Clusters." *Proceedings of the VLDB Endowment*, 2013.