Evaluating the Combined Impact of Node Architecture and Cloud Workload Characteristics on Network Traffic and Performance/Cost

Diman Zad Tootaghaj[1], Farshid Farhat[1], Mohammad Arjomand[1], Paolo Faraboschi[2]
Mahmut Taylan Kandemir[1], Anand Sivasubramaniam[1], Chita R. Das[1]
[1]The Pennsylvania State University, [2]Hewlett-Packard Labs
{*dxz149, fuf111, arjomand, kandemir, das, anand*}@*cse.psu.edu*, {paolo.faraboschi}@hp.com

Abstract—**The combined impact of node architecture and workload characteristics on off-chip network traffic with performance/cost analysis has not been investigated before in the context of emerging cloud applications. Motivated by this observation, this paper performs a thorough characterization of twelve cloud workloads using a full-system datacenter simulation infrastructure. We first study the inherent network characteristics of emerging cloud applications including message inter-arrival times, packet sizes, inter-node communication overhead, self-similarity, and traffic volume. Then, we study the effect of hardware architectural metrics on network traffic. Our experimental analysis reveals that (1) the message arrival times and packet-size distributions exhibit variances across different cloud applications; (2) the inter-arrival times imply a large amount of self-similarity as the number of nodes increase; (3) the node architecture can play a significant role in shaping the overall network traffic; and finally, (4) the applications we study can be broadly divided into those which perform better in a scale-out or scale-up configuration at node level and into two categories, namely, those that have long-duration, low-burst flows and those that have short-duration, high-burst flows. Using the results of (3) and (4), the paper discusses the performance/cost trade-offs for scale-out and scale-up approaches and proposes an analytical model that can be used to predict the communication and computation demand for different configurations. It is shown that the difference between two different node architecture's performance per dollar cost (under same number of cores system wide) can be as high as 154 percent which disclose the need for accurate characterization of cloud applications before wasting the precious cloud resources by allocating wrong architecture. The results of this study can be used for system modeling, capacity planning and managing heterogeneous resources for large-scale system designs.**

Keywords—Workload characteristics; Performance/Cost analysis

I. Introduction

Cloud applications, ranging from interactive query-based jobs to high performance computing applications are driving the development of current datacenters that are composed of tens of thousands of nodes to handle extremely large amounts of data processing and operation execution. Efficiently executing these applications requires having sufficient computational, storage, and network bandwidth resources. For example, many Map-Reduce jobs, have bursty network communications during the map and data shuffling phases [1, 2, 3, 4, 5]. Similarly, many cloud applications and high-performance computing (HPC) applications need high communication bandwidth [6].

While every other aspect of datacenter performance have been improved drastically, network bandwidth and latency has been a source of performance degradation in cloud computing for years [6, 7]. High network latency abandons any hope for getting performance benefit on large number of machines and makes the scale-out approach [8, 9] challenging for network-intensive applications. On the other hand, the ever-growing data sizes in *Big Data* era, that cannot be processed in a single server or reside in memory makes distributed processing unavoidable. Therefore, given a workload and a set of computational and storage resources, an in-depth study of network traffic demand, can be the key to reducing costs and boosting performance. Analyzing the combined impact of hardware architecture and workload can help cloud providers in designing performance/cost efficient datacenters. There exists prior works [10, 11, 12] that studied workload characteristics and network traffic in datacenters. High-performance computing (HPC) nodes produce higher traffic bursts compared to low-performance computing nodes and on the other hand can handle part of traffic within cores using inter-core interconnection network. In addition to node architecture, the other factor that shapes message traffic is the inherent temporal characteristics of the workload. Most of the previous studies on network topology exploration employ either synthetic traffic or non-real workloads [13, 14, 15]. However, the hardware architectures employed by datacenters and workloads are changing rapidly; as a result, the traffic patterns and application characteristics taken from prior studies may not be applicable to current systems. To the best of our knowledge, none of these studies investigated the impact of node architecture and real workload characteristics on inter-node network traffic.

In addition, conventional trend in academia and industry suggests using a cluster of commodity servers, which is called *scale-out* for cloud computing [8]. Appuswamy *et al.* [16] suggest adding more resources to a single server, which is called *scale-up* approach. In this paper, we discuss performance/cost trade-offs for these controversial approaches for different cloud applications. We consider different types of cloud workloads ranging from those that represent Map-Reduce and client-server paradigms in *CloudSuite* [17, 18, 19] to memory-intensive large-scale scientific computing benchmarks in *Mantevo* [20], and network-intensive parallel graph algorithms in *Graphlab* [21, 22].
The main **goal** of this paper is to study the *combined impact* of node architecture (the number of cores, intra-node network, cache/memory hierarchy) and cloud workload characteristics on inter-node network traffic as well as performance/cost analysis. Using a set of twelve cloud workloads with different characteristics, the specific research questions that we strive to answer can be summarized as follows:

- What is the message inter-arrival times' distribution and packet size distribution of different cloud workloads to regenerate the same traffic pattern?
- What factors affect the burstiness of network traffic in the cloud? How much role does the node architecture play in order to design better network buffer managements?
- What is the ratio of communication time over computation time for cloud workloads and how does it scale with respect to the volume of data to provision network capacity for each cloud workload?
- How does the bandwidth requirements of different cloud benchmarks change with respect to the basic node parameters such as the number of cores and cache/memory capacity? This observation helps a designer to provide enough network resources when configuring resources to the current datacenter architecture.
- How much is the performance/cost of different workloads for scale-out or scale-up approaches? which helps us to find the right way to expand the datacenter for the running workloads.
- What is the impact of node architecture on network traffic and performance/cost? Which of scale-out or scale-up approaches is better for each studied cloud workloads?

We carried out a series of experimental study using a full-system simulation that gives us exploring the effect of hardware parameters on the system performance. It also helps to isolate traffic pattern of different applications [23]. We used a set of modern workloads in cloud study including *Cloudsuite* [17, 18, 19], *Mantevo* [20] and *Graphlab* [21, 22] benchmark suites. We also generalized experimental study for large-sized datacenters to ease capacity planning and performance optimization. We can derive the following results from our experimental study:

- The studied workloads can be categorized into three different groups: memory-intensive applications, network bandwidth-intensive applications, and network latency-sensitive applications. Applications belonging to different groups require different design optimization considerations.
- Running network bandwidth-intensive or latency-sensitive applications on a scale-out hardware architecture does not give us much performance/cost benefit compared to the scale-up approach. On the other side, memory-intensive applications scale well and the scale-out approach gives better performance/cost.
- For most of the studied cloud benchmarks, the packet inter-arrival times follow *lognormal* or *extreme value* distribution, which is a long-tailed distribution and causes self-similarity in network traffic. Self-similar traffic increases queue length in the network and this degrades the performance of the system. There exist prior works on how to perform buffer management in the presence of self-similarity in network traffic [24]. Self-similar traffic is quantified by a *Hurst* parameter. Larger *Hurst* parameter represents higher self-similarity for a flow and more possibility to congest the network. Consequently, if we knew the self-similarity quantity of each application in advance, it would be possible to employ better buffer management and achieve better performance. It was observed that increasing the dataset size and the number of nodes increase the hurst parameter for self-similarity.
- We used the parameters of AMD Opteron server processors [25, 26] and observed that the node architecture (e.g., number of cores, cache/memory capacity) plays an important role on bandwidth; for instance, the effect of node configuration (under the same number of cores system wide) can be as much as 40.7% on execution time and 154% on performance per dollar; and the effect of network configurations on execution time can be as much as 74%.

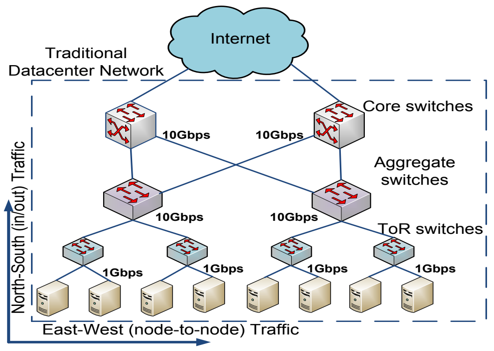


Fig. 1: A three-layer datacenter architecture.

The rest of this paper is structured as follows. Section II discusses the motivation and background behind this work. Section III explains experimental setup. Section IV studies network characteristics of studied workloads. Section V describes the evaluation results. Section VI summarizes the potential implications of our major findings. Section VII studies related work and finally Section VIII concludes the paper.

II. Motivation and Background

A. Motivation

With the rapid increase in consumer demands and increasing complexity of the cloud applications, datacenter designers are steadily revisiting design of nodes and network architectures in datacenters. As both applications and node architectures are rapidly changing, analyzing the impact of both on network traffic is becoming increasingly important. Large cloud computing vendors such as Facebook, Amazon, and Google rely heavily on traditional network structures, which consist of two or three levels of switch hierarchy shown in Figure 1 [10, 14].

As technology improves, more and more cores are being embedded into a single node and the communication bandwidth requirement of datacenters keeps increasing. As pointed out by prior studies [6, 7, 13], one of the main limitations of today's large-scale computing is the network communication latency and bandwidth. There have been several prior studies attempted to find better network topologies and routing algorithms in datacenters [13, 14]. However, to our knowledge, there is no prior work that characterizes the combined impact of machine architecture/parameters and cloud application characteristics on the network traffic, performance/cost of scalability. Such a study could help us identify the application's specific bottlenecks and ultimately reach cost-efficient datacenter designs for different characteristics of cloud workloads. In addition, accurate workload characterization is important for system modeling and capacity planning. Specifically, the capacity planning for large-scale system design needs more accurate modeling and understanding of workload requirement and behavior. Operating and running large computation in large scale is expensive and large cloud vendors have to know how to expand node architecture and network architecture resources to satisfy the ever-increasing job workload demands. Recent trend in academia and industry is to use large number of commodity servers for cloud computing [8, 9]. However, based on network communication pattern, different workloads have completely different performance metrics for scale-up and scale-out approach. Therefore, in this paper, we conduct a simulation-based study, using a full-system simulator, that simplifies design space exploration, to perform such evaluations in smaller scale to find out the network traffic pattern of these applications in small-scaled systems and generalize it to large-scale ones. To this end, we derive an analytical model and use non-linear regression analysis to find the accuracy of the predicted model.

B. Background

Traditional datacenter architectures consist of two or three layers structured as a tree, where higher layer switches have high capacity with more ports but more expensive. In order to reduce the cost, most datacenter network designers use over-subscription factor of 2.5:1 to 8:1 [13]. An over-subscription of n:1 means that in worst case traffic patterns, only 1/n of the total communication bandwidth is available to different nodes in that traffic pattern. The concurrent flows for each cloud application plays an important role on how much we can over-subscribe a network architecture. For example, if an application has all-to-all communication pattern, oversubscribing the network would degrade the performance since the network would be the bottleneck resource. Concurrent flow analysis can guide system administrative on choosing the appropriate factor for over-subscription. Figure 1 shows an example of a three-layer datacenter network architecture, where the servers at each rack are connected to the Top of Rack (ToR) switches with 1Gbps Ethernet links. ToR switches are connected to the aggregate switches using 10 Gbps links and finally aggregate switches are connected to the core switches using a fat-tree topology. It is important to provision enough bandwidth for each link in the hierarchy when we scale-up or scale-out the servers. Network traffic can be distinguished as either node-to-node (East-West) communication or in-out traffic (North-South). North-South traffic patterns happen in client-server applications, where the clients from the internet side (Figure 1) send queries to the servers and get responses. The communication pattern in North-South traffic patterns is mainly between client and servers and the servers do not have communication among themselves. While, in East-West traffic patterns, the communication is mainly among servers. The client-server and interactive query-base applications in *Cloudsuite* have North-South traffic pattern and the servers never communicate with each other. However, most scientific benchmarks like graph algorithms from *Graphlab* and *Mantevo* benchmarks in our experiments have East-West traffic behavior.

Provisioning for East-West traffic is more difficult than North-South traffic, because North-South traffic challenges can be solved by means of load balancing and replication techniques. However, it has been extensively discussed that East-West network bandwidth and latency is the main source of performance degradation in large-scale computing [6]. Therefore, we focus on East-West traffic patterns as dominant behavior in datacenters.

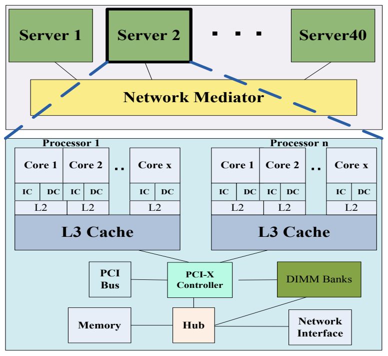


Fig. 2: High-level view of the simulated architecture.

TABLE I: Node parameters and their default values.

Processor parameters and their default values	
Clock frequency	2400MHz
Main Memory latency	150cycles
L1 caches	Split I and D, 64kB private, 2-way, 64B, LRU, write-through, 2-cycle hit
L1 TLB	Split I and D, 4kB private, 32 entries, 2-cycle latency
L2 cache	Unified, 1MB private, inclusive, 4-way, 64B, LRU, write-back, 10-cycle hit
L3 cache	4MB shared, NUCA, inclusive, 16-way, 64B, LRU, write-back, 4-cycle tag, 10-cycle data hit, 40-cycle CPU to L2

III. Experimental Setup

In this section, we describe the simulation setup and different node and network architectures that we tested. To capture fine-grained network traffic pattern (in the granularity of micro second) for any specific workload, we employed COTSon full-system simulator [27, 28], that is based on AMD Simnow [29]. Simnow simulates an entire server machine including cores, memory, I/O and network interfaces. To simulate multiple nodes, COTSon provides networking for Simnow and uses a parallel discrete-event model, allowing the simulation of multi-core clusters. The simulator uses a mediator to provide a network interface between nodes (or servers) [30], and it uses Ubuntu 64-bit x-86 3.5.0-23-generic Linux as the operating system.

A. Simulated Configurations

1) Node Architectures: We used the parameters of an AMD Opteron server processor as a baseline in our simulations [25]. Table I gives the main architectural parameters of the baseline system configuration. Each core has its own private instruction and data L1 and L2 caches and an L3 cache is shared among all the cores in the processor. The memory, L3 cache, network interface and DIMM banks are connected through a hub device [31, 32, 33]. Figure 2 shows the target architecture simulated in our experiments. Table II shows the different node architectures varied in this work. We changed the number of servers from 1 to 40 (scale-out degree) and the number of cores per node from 1 to 6 (scale-up degree).

2) Network Architectures: We used three different network architectures with different bandwidth, latency and congestion parameter, as shown in Table III. We used the standard ToR switches that contain 48 GigE ports. This table specifies each network configuration in terms of maximum available bandwidth of links (in Gbps), minimum latency of the switches (in μs) and a congestion factor (*cfactor*) that shows the queuing latency of the switches.

TABLE II: Different node architectures considered in this work.

Architecture	L3 size/latency	Memory Size	cores
Default	4MB/40cycles	512M	1
Double Cache	8MB/60cycles	512M	1
Double memory	4MB/40cycles	2048M	1
Dual-core	8MB/60cycles	1024M	2
4-core	16MB /80cycles	2024M	4
6-core	24MB/100cycles	6144M	6

TABLE III: Different network architectures.

Network Parameter	Network 1	Network 2	Network 3
Ports	48*1 GbE	48*1 GbE	48*1/10 GbE
Maximum Bandwidth	1 Gbps	1 Gbps	100 Mbps
Latency	$4\mu s$	$40\mu s$	$4\mu s$
cfactor	2	10	10

B. Evaluated Workloads

We used a diverse set of cloud applications ranging from interactive query-based to high-performance scientific workloads. The workload set include seven programs from *Graphlab* benchmark suite [21], two programs from *Cloudsuite* benchmarks [17], and three programs from *Mantevo* benchmarks [20]. The twelve benchmark programs used in this work and short descriptions can be summarized in Table IV[1]. The graph analytic toolkit from *Graphlab* has different applications that analyze a graph structure. As input for these applications, we used Stanford large network dataset collections [34] which include different graphs from social networks and communication networks. We also used two benchmarks from *Cloudsuite*, that are based on online applications. The *Graphlab* uses Open MPI and *Hadoop* implementation of Map-Reduce to perform graph computations on large graphs. The *Mantevo* benchmarks are open-source mini-applications to analyze, predict and improve HPC systems.

C. Evaluation Methodology

To quantify the impact of node and network architectures on the network traffic pattern and overall workload performance, we evaluate five different scenarios:

1) Changing the number of cores per socket, and increasing the total cache/memory capacity, we evaluate the effect of scale-up on network communication as well as performance/cost trade-offs.
2) Changing the capacity of LLC per node, we determine the memory-intensity of the applications.
3) Changing the total number of nodes in the system, we evaluate the network traffic pattern as well as the traffic self similarity metrics.
4) Changing the network bandwidth and latency, we determine the sensitivity of the applications to the network configurations.
5) We compare the scale-out of 40 single-core machines with the scale-up of 12 quad-core machines for different applications.

[1]The names in parentheses are abbreviations used in this paper

D. Metrics

The metrics used to evaluate the impact of node configuration and workload characteristics on network traffic are:

- **Communication over computation overhead** gives the overheads incurred due to parallel execution. This metric can be used to evaluate the communication overhead experienced when running a given benchmark in parallel.
- **Packet inter-arrival times and packet size distribution** These metrics, capture the burstiness of the incoming traffic and if known by a designer, he/she can regenerate the traffic pattern for synthetic traffic generation. Furthermore, knowing these metrics in advance help us to provision network buffers and have better bandwidth allocation for different applications.
- **Self similarity** indicates the burstiness of the network traffic for different benchmarks.
- **Bandwidth requirement** gives the amount of bandwidth needed between different nodes for a given dataset and hardware architecture.
- **Concurrent flows** shows the maximum number of concurrent flows in an application/workload during execution. This metric can be used to predict the required bisection bandwidth of each application to choose appropriate over-subscription factor. For example, if an application has an all-to-all traffic pattern (highest number of the concurrent flows), over-subscribing the network degrades the overall system performance.
- **Performance per dollar cost** indicates how much performance increase we will get for each dollar we spend on system configuration.

IV. Network Characteristics' of Applications

A. Inter-arrival Times and Self Similarity

Most prior works on datacenter networks use synthetic traffic with exponential distribution to tune network and/or application parameters. However, in our experiments the inter-arrival time of our cloud applications follow a long-tailed distribution [37], that can result in performance degradation, unless we provision enough buffers in the network.

We found that, for most of the evaluated graph analytic benchmarks, the *generalized extreme value* distribution [2] gives the best fit for packet inter-arrival times' distribution. We further observed that, the *lognormal* distribution, which has two degrees of freedom, seems to be a good fit for *Graphlab* benchmarks. Table VI lists the parameters for each distribution using the maximum likelihood estimation and the normalized Euclidean distance (error) we obtain for each distribution.

Similar to our finding, prior studies [38, 39] have shown that the traffic pattern of Ethernet follows a self-similar distribution which degrades network performance. Self-similar processes can be described using long-tailed distributions like *Pareto*, *lognormal*, and *Weibull* distribution. Self-similarity in networks is not a pleasant phenomenon, since network performance can degrade with increasing self-similarity due to, primarily, the queue length increase. The effect of self similarity on

[2]Generalized extreme value distribution has three degrees of freedom and can result a better fit with less error

TABLE IV: Evaluated workloads.

Application	Description
pagerank (pgrnk)	An algorithm used by Google that computes the pagerank of each vertex in the graph.
format convert (fcvt)	Converts a graph from a specific format, for example snap, tsv, adj or binsv4 to another format.
undirected triangle count (udtc)	Counts the total number of triangles in a graph using the algorithm in [35].
directed triangle count (dtc)	Counts the total number of directed triangles in the graph.
kcore decomposition (kcore)	Iteratively computes sub graphs of k cores in a given graph. A sub graph is called k-core if and only if all vertices in the sub graph is at least of degree k.
connected components (cc)	Computes all connected components and the number of vertices in that component for a given graph.
approximate diameter (apxr)	Computes the approximate diameter of a given graph using the proposed algorithm in [36].
Data Serving (dsrv)	A benchmark from Cloudsuite [19] for data store systems that is used for large-scale web applications.
memcached (mem)	A client-server architecture for distributed memory caching. The servers keep key-value stores and clients send query for these key-values. The keys are at most 250 bytes and values are at most 1MB.
CoMD	A part of Mantevo benchmarks and is an mpi implementation of molecular dynamics algorithms which is used in material science.
MiniFE	An approximation to an unstructured implicit finite element code. The application was configured with MPI support.
HPCCG	Implements a *Conjugate Gradient* solver, where the coefficient matrix is stored in a sparse matrix format.

TABLE V: Hurst parameter estimation for different number of nodes.

Method / **# nodes**	**aggvar**	**boxper**	**diffvar**	**peng**	**per**	**R/S**
5	0.478	0.454	0.253	0.448	0.430	0.571
10	0.484	0.461	0.388	0.485	0.441	0.577
15	0.541	0.472	0.453	0.580	0.453	0.601
20	0.584	0.480	0.666	0.597	0.462	0.653
40	0.634	0.520	0.712	0.621	0.482	0.664

network performance is studied by Park *et al.* [40]. Taqqu *et al.* [37] proposed several strategies to estimate the self similarity parameter. We follow [37] and evaluate Hurst parameter as an estimation of self similarity. Table V gives the result of this estimation using the different strategies discussed in [37]. It is observed that increasing the number of nodes increases the self-similarities. Consequently, network designer need to provision more buffers when running the workloads on large number of machines. Those flows, which have higher Hurst parameter, fill up the networks buffers and non-self similar traffics would be discarded. By knowing the Hurst parameter for different flows, a designer can estimate queue length for network buffers and potentially design better buffer management system.

B. Packet Size Distribution

In addition to the message inter-arrival times, packet size distribution has to be known for regeneration of the same traffic pattern. For instance, one needs to consider a higher network bandwidth and more buffers in networks with larger packet sizes. Here, we report the analysis results of packet size distribution of our benchmarks. The simulation results, in Figure 3 show that packet size distribution for the benchmarks which use Map-Reduce framework follow a bimodal distribution with two peaks in 66 bytes and 1514 bytes, that is the maximum Ethernet packet size. *Pagerank*, *approximate diameter*, *connected component*, *format convert*, *directed triangle count* and *undirected triangle count* are the applications, which have larger packet sizes with a bimodal distribution. However, for web applications, packet sizes have a multi-modal distribution with smaller packets. *Memcached*, *data serving* and *kcore* are three applications that have a multi-modal packet size distribution with fewer packet sizes. Knowing the packet size distribution of these workloads helps us to perform better traffic and quality of service (QoS) management in the network only by extracting the packet size in the header. Smaller packets can get higher priority than larger packets. Also the high frequency of large packets (1514 bytes) shows the room for improving the packet control protocol in current datacenter networks to allow transmitting higher sizes of packets.

C. Concurrent Flow Analysis

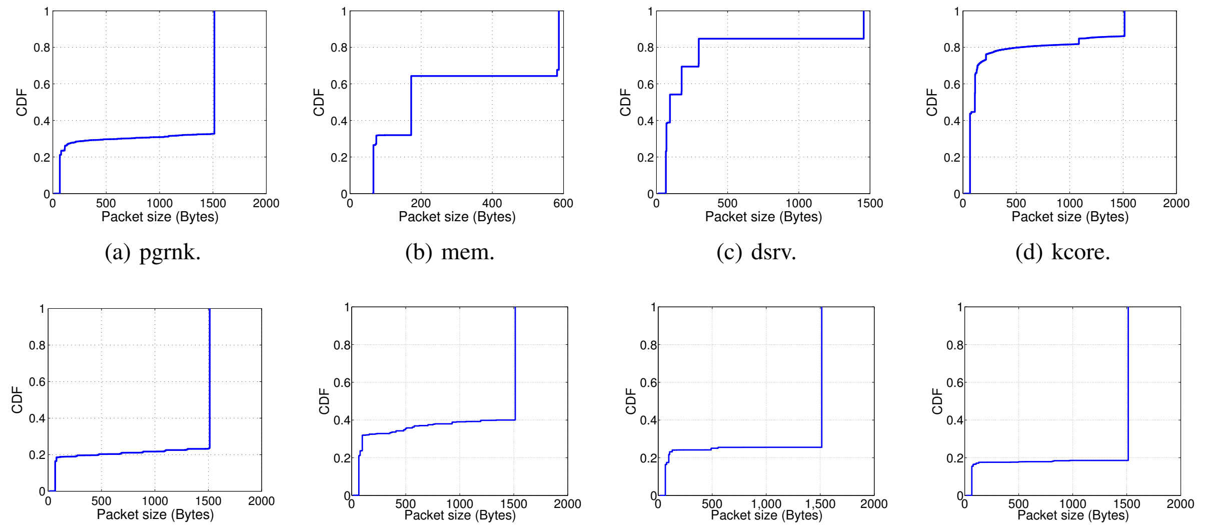
Different applications generate different ranges of long-term or short-term traffic flows. Datacenter networks should be able to tolerate large traffic bursts, provide low latency for short flows and high utilization for long flows. In addition, the number of concurrent flows in an application gives us a metric to determine how much the bisection bandwidth is being utilized in the network. Computing the number of concurrent flows for an application helps us to have a better understanding of the bisection bandwidth utilization of that application. This helps an administrator to determine how to over-subscribe the network for cost reduction. To compute concurrent flows of each application, we use the steps given in Algorithm 1. Using the time and size of packets exchanging between two nodes, Algorithm 1 finds all the longest contiguous intervals that these two nodes send receive. The threshold of having a connection is the mean rate of all communications, i.e., if packet size per its inter-arrival time is higher than the mean (threshold), that interval is considered as a *flow* between these two nodes.

Table VII shows the comparison of concurrent flows from the master node to all slave nodes for each application running on a 10 nodes. We observed that, *kcore*, *Data serving*, and *directed triangle count* have the most concurrent flows. Therefore, running these applications on the same rack can be expected to reduce congestion in the higher layers of the datacenter hierarchy. In addition, we compute average flow duration of different benchmarks, some benchmarks, like *kcore*, *data serving*, and *memcached*, have long-term flow duration which makes them suitable fit for hybrid architecture (circuit and packet switching) networks where long-term duration flows can go through the circuit switching network. Other benchmarks, with short-term flows, need to go through the packet switching network part of the hybrid network.

V. Evaluation

A. Communication over Computation Overhead

We start by studying the communication overhead with different number of nodes. We change the number of nodes from 1 to 40 and measure the total execution time it takes for the server nodes to perform *pagerank* graph computation. We chose *pagerank* benchmark as a representative of graph



(a) pgrnk. (b) mem. (c) dsrv. (d) kcore.

(e) apxr, cc, fcvt, dtc, udtc. (f) MiniFE. (g) HPCCG. (h) CoMD.

Fig. 3: CDF of packet sizes in the graph analytic and web benchmarks.

TABLE VI: Error and best fitting parameters of different distributions for packet inter-arrival times in different benchmarks.

Application \ Distribution	exponential	gamma	gev	weibull	lognormal
Pagerank	Error: 0.030 Mean: 0.6711	Error: 0.0058 Shape: 0.1516 Scale: 0.435	Error: 9.5e-4 Shape: 1.85, Scale: 2.76e-5 Location: 1.47e-5	Error: 7.1e-4 Scale: 1.51e-4 Shape: 0.34	**Error: 5.7e-4 Mean: -10.0828 SD: 2.39**
connected components	Error: 0.0386 Mean: 0.0086	Error: 0.0063 Shape: 0.138 Scale: 0.062	**Error: 5.4e-4 Shape: 1.397, Scale: 1.93e-5 Location: 1.30e-5**	Error: 8.6e-4 Scale: 1.05e-4 Shape: 0.32	Error: 6.1e-4 Mean: -10.4062 SD: 2.20
data serving	Error: 9.3404 Mean: 1.91	Error: 0.0308 Shape: 0.158 Scale: 12.07	**Error: 0.1240 Shape: 4.43, Scale: 0.459 Location: 0.1035**	Error: 0.0286 Scale: 0.347 Shape: 0.213	**Error: 0.0141 Mean: -4.178 SD: 8.40**
kcore	Error: 0.0079 Mean: 0.0028	Error: 0.0045 Shape: 0.397 Scale: 0.007	**Error: 5.7e-4 Shape: 0.805, Scale: 0.00047 Location: 0.00039**	Error: 0.0019 Scale: 0.0012 Shape: 0.59	Error: 0.0013 Mean: -7.54 SD: 1.69
memcached	Error: 9.9728 Mean: 2.037	Error: 0.0214 Shape: 0.107 Scale: 19.08	**Error: 0.0096 Shape: 7.76, Scale: 9.27e-5 Location: 1.29e-5**	Error: 0.0156 Scale: 0.0303 Shape: 0.163	Error: 0.0125 Mean: -6.834 SD: 7.94
directed triangle count	Error: 0.0282 Mean: 0.0063	Error: 0.0059 Shape: 0.1386 Scale: 0.0456	**Error: 4.9e-4 Shape: 1.12, Scale: 1.50e-5 Location: 1.18e-5**	Error: 8e-4 Scale: 7.117e-5 Shape: 0.34	**Error: 5.5e-4 Mean: -10.672 SD: 1.94**
undirected triangle count	Error: 0.0345 Mean: 0.0077	Error: 0.0062 Shape: 0.138 Scale: 0.055	**Error: 6.9e-4 Shape: 1.101, Scale: 1.62e-5 Location: 1.37e-5**	Error: 8.7e-4 Scale: 8.76e-5 Shape: 0.34	**Error: 7.1e-4 Mean: -10.49 SD: 1.97**
approximate diameter	Error: 0.0661 Mean: 0.014	Error: 0.0072 Shape: 0.127 Scale: 0.112	**Error: 7.2e-4 Shape: 1.088, Scale: 1.60e-5 Location: 1.39e-5**	Error: 9.7e-4 Scale: 9.26e-5 Shape: 0.32	Error: 7.3e-4 Mean: -10.473 SD: 2.00
format convert	Error: 0.0696 Mean: 0.015	Error: 0.0073 Shape: 0.123 Scale: 0.121	**Error: 6.5e-4 Shape: 1.026, Scale: 1.43e-5 Location: 1.26e-5**	Error: 9.5e-4 Scale: 7.77e-5 Shape: 0.32	Error: 6.8e-4 Mean: -10.62 SD: 1.94
CoMD	Error: 0.4073 Mean: 0.00239	Error: 0.2204 Shape: 0.152 Scale: 0.0157	Error: 1.31e-2 Shape: 0.7142, Scale: 1.0832e-5 Location: 1.079e-5	Error: 1.57e-2 Scale: 3.72e-5 Shape: 0.3955	**Error: 1.10e-2 Mean: -11.08 SD: 1.67**
MiniFE	Error: 0.5245 Mean: 0.00389	Error: 0.2084 Shape: 0.140 Scale: 0.0277	Error: 6.5e-2 Shape: 1.2634, Scale: 1.048e-5 Location: 5.476e-6	Error: 3.41e-2 Scale: 7.04e-5 Shape: 0.305	**Error: 3.37e-2 Mean: -11.07 SD: 2.91**
HPCCG	Error: 0.444 Mean: 0.0029	Error: 0.2100 Shape: 0.154 Scale: 0.0188	Error: 4.4e-2 Shape: 1.068, Scale: 1.4731e-5 Location: 1.1441e-5	Error: 3.58e-2 Scale: 6.70e-5 Shape: 0.345	**Error: 3.91e-2 Mean: -10.80 SD: 2.17**

applications, because it has more inter-node communication overhead. We recall that that the total execution time taken by the application to perform the required computation on n nodes can be expressed as:

$$\text{Total execution time} = S + P/n + O_n, \tag{1}$$

where S is the sequential part of the program, P is the parallelable part, and O_n is the communication overhead. There exists a trade-off between the degree of parallelism (n) and the communication overhead (O_n) incurred due to parallelism. As we increase n, the program runs more parallel but on the other hand we experience more communication overhead due to the required synchronization to merge the processed data. Figure 4 plots the execution time versus the number of nodes for the application under study (*pagerank*) under different dataset sizes. It can be seen that, in general,

Algorithm 1: Computing concurrent flows.

```
Input: logs of all nodes communications including packet time and size.
Output: Number of concurrent flows.
1 Partition all logs from a source (S) to any destination (D).
2 Get the inter-arrival times (IAT) vector of the logs in 1 with packet size (PS) vector.
3 Compute flow rate (FR) vector from S to any D with zero-division protection as $FR[i] = PS[i]/(IAT[i] + 1)$ $i = 1..n$.
4 Compute mean of flow rate vector (MFR) as a threshold.
5 For all D, flow from S to D (F(S,D)) exists if its FR[i] is higher than MFR.
6 At any time for all D, flow duration from S to any D (FD(S)) is sum of the $IAT[i]$ $i = 1..n$ which its $F(S, D)$ exists.
7 Repeat 1 for any other source.
```

TABLE VII: The comparative time analysis of the maximum number of concurrent flows for studied workloads.

Application	Concurrent Flows
Data serving	71
approximate diameter	20
connected component	42
directed triangle count	54
format convert	18
kcore	75
pagerank	28
undirected triangle count	45
memcached	1
CoMD	40
MiniFE	10
HPCCG	5

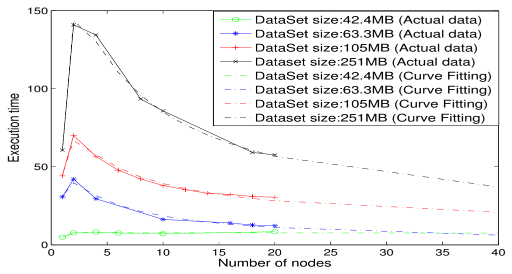


Fig. 4: Execution time versus the number of nodes for *pagerank* using default server architecture (architecture 1 in Table II).

as we increase the size of the dataset, the performance keeps increasing for larger number of nodes. However, the overhead of communication is too much for small dataset sizes. For dataset sizes less than 60MB, as we increase the number of nodes, no performance improvement is achieved and there is no point in increasing the scale-out degree. Further, increasing the number of nodes (servers) from 1 to 2 degrades performance as the impact of the overhead is typically more pronounced with the small number of nodes. Using non-linear regression methods, the communication overhead in Equation 1 can be modeled as follows:

$$O_n = (\alpha_1 + \alpha_2/n) * log(n). \tag{2}$$

where α_1 represents the coefficient of the communication overhead of the sequential part (when one server forks the tasks for the other parallel servers), and α_2 represents the coefficient of the communication overhead of the parallel part (when parallel servers want to join the processed data in one place). It is known that the implementation of the program has a logarithmic overhead [41]. Therefore, Equation 2 can accurately predict the execution time with respect to the datasize and the degree of parallelism. For small dataset sizes, the value of α_1 is

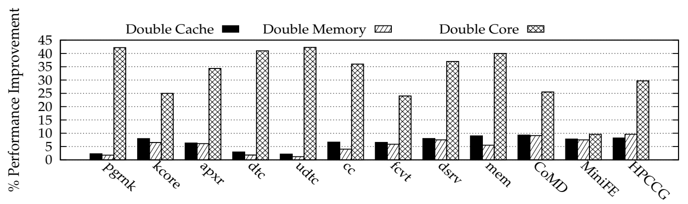


Fig. 5: Performance improvement of three different configurations with respect to the baseline.

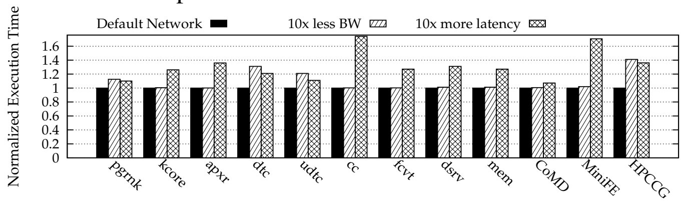


Fig. 6: Normalized execution time of three different network configurations with respect to the baseline.

very close to zero and increases as we increase the dataset size. In addition, it is observed that, running the same dataset size on very large number of nodes increases the communication overhead such that beyond a certain point, parallelism level saturates and the communication overhead keeps increasing. Figure 4 plots the result of curve fitting and predicted tail (using dashed lines) using this model for *pagerank* and for different data sizes. The experiments in this figure show that, when doubling the dataset size, the overhead constant, α_2, increases by 178%; however, the parallel constant, P, in Equation 1, only increases by 50%.

B. Performance Comparison of Different Architectures

To explore the impact of different node architectures, we evaluated the performance of different applications using the studied configurations, as was shown in Tables II and III, respectively. Figure 5 shows the performance improvement that each configuration gets with respect to the baseline system for different applications. The performance improvement is the average performance improvement for 10 different dataset sizes. It is observed that *pagerank*, *approximate diameter*, *directed triangle count*, *undirected triangle count* and *connected components* are applications which are less cache sensitive and get more performance improvement with more number of cores per server. On the other hand, *kcore*, *CoMD*, *MiniFE* and *HPCCG* are more cache and memory sensitive and increasing the number of cores per server increases the contention for memory bandwidth. Therefore, these benchmarks show less performance improvement with more number of cores per server.

Further, based on the results of experiments with different network architectures, we can divide these applications into network bandwidth-intensive and network latency-sensitive categories. Figure 6 shows normalized execution time of the applications with different network configurations with respect to the baseline. It is observed that, *MiniFE*, *HPCCG*, *approximate diameter* and *connected components* are applications which are more sensitive to the network latency. On the other hand, *pagerank*, *directed triangle count* and *undirected triangle count* are bandwidth-intensive applications. Later, in section V-D, we show that the applications can have different performance/cost benefits in a scale-out or scale-up approach based on how network sensitive they are.

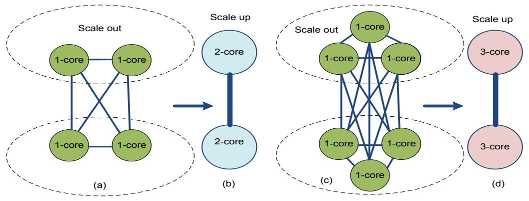


Fig. 7: Scale-up approach with two and three cores per node versus scale-out approach.

C. Bandwidth Comparison between Scale-up and Scale-out

In this part, we quantify the effect of different node architectures on the bandwidth requirement of our workloads. In particular, we target iso-core architectures, and compare the scale-up and scale-out approaches. Figures 7(a-b) and 7(c-d) show 4 cores and 6 cores in the scale-out and scale-up configurations, respectively. Let us assume that a scale-up node is equal to packing k cores in scale-out to construct a k-core scale-up machine as shown in Figure 7. The virtual links in this figure show the communication demand between any two nodes in scale-out or scale-up configuration. In real configurations, nodes are not connected in a fully connected graphs, but the average bandwidth between two different nodes does not change. Let S be the subset of links in the scale-out approach which connects the two parts of the graph, which are forming one vertical link in the scale-up approach. Investigating the different scale-out and scale-up approach with different number of nodes and different dataset sizes, we observed that if we aggregate k nodes together the average communication bandwidth between any two k-core machines is approximately:

$$\text{Avg BW in k-core scale up} = k^2 * \frac{k^2}{\binom{2k}{2}} * \text{Avg BW in scale-out} \quad (3)$$

In addition, the maximum bandwidth of each link in scale-up approach can be computed as follows:

$$max(BW_{\text{k-core scale-up}}) = max(\sum_{i=1, i \in S}^{k} BW_{\text{link \#i in scale-out}}) \quad (4)$$

Figure 7 plots virtual links between any two communicating nodes in a scale-up approach which is k^2 times more than each link in the scale-out approach [3]. Using the proposed analytical approach, a designer can predict how much to increase the average bandwidth of network when scaling up server nodes. Since the maximum bandwidth in today's datacenter switches is 10Gbps, scaling up the servers at some point may lead the network bandwidth be the bottleneck in the system. Figure 8 shows the average bandwidth usage of different links in scale-out and scale-up approach for applications with east to west traffic patterns. Our experiments also show that the web applications where a client sends request to servers, utilize the high level switches, regardless of mapping of cores to nodes, since there is very little communication among servers. However, in

[3]The formula is derived using the fact that the average communication of each k-core machine should be approximately k^2 times more than the scale-out approach, however the total communication bandwidth is reduced by a factor of $\frac{k^2}{\binom{2k}{2}}$ which is the total number of outgoing links in scale-up approach over the total number of outgoing links in the scale-out approach. For example in a 2-core scale-up approach in Figure 3, every 4 link construct one link between two nodes in scale-up, and the total amount of off-chip communication is reduced by $4/6$ and in a 3-core scale-up approach every 9 links in the scale-out construct one link in scale-up and the total off-chip communication is reduced by $9/15$.

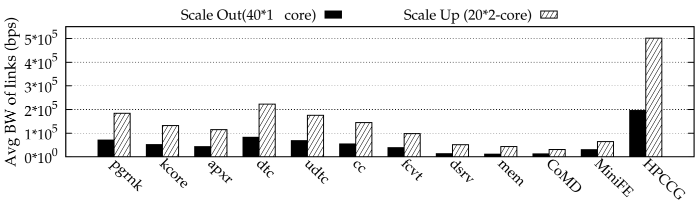


Fig. 8: Average bandwidth usage of different benchmarks for scale-up and scale-out approach.

Map-reduce benchmarks, the traffic volume between different servers is very high; consequently, it would be better, if possible, to use the servers in the same rack to avoid traffic to be directed to high level switches.

D. Performance Cost Analysis

In this section, we evaluate performance/cost analysis of different applications in a scale-out and scale-up approach. To estimate the cost for datacenter in scale-out and scale-up approach, we assume a power usage effectiveness (PUE) of 1.7, and utility price of 0.07 dollar per kwh. As shown in Figure 5, most of cloud computing workloads are CPU and memory intensive. Thus, we assume CPU and memory price to be the major contributor of capital costs and power cost to be the major contributor of operating cost [4] [16, 42, 43].

Figure 9 shows the total cost of ownership for different configurations with 192-cores in a scale-out or scale-up approach, using the offered cost of simulated AMD processors from [26]. For each configuration in Figure 9, represents the number of nodes, B refers to the total number of cores inside a single node and C is the number of sockets for each node. From total cost of ownership (TCO) perspective, it is observed that 16×12-core dual socket servers has the optimal TCO for the servers in a datacenter, but TCO does not consider workload's performance. Thus, we came up with a better metric to include TCO and performance together. Figure 10 shows speed up/cost with respect to one single core machine for different applications for scale-out and scale-up approaches. We assume iso-core, iso-cache capacity and iso-memory configurations. As it is seen in the figure, graph applications have little scalability and get better performance in scale-up approach. In fact, scale-up approach has better performance/cost for graph applications that needs to access random bits of data frequently. These applications are network bandwidth-intensive and the communication overhead of scale-out is abandons any hope to get the same performance as in the scale-up approach. But HPCCG as a bandwidth-intensive, latency-sensitive, and memory-intensive workload performs better in a scale-out approach. However, applications with high locality benefit from a scale-out approach. The total aggregated memory bandwidth of scale-out approach is higher than the scale-up, so memory intensive applications get more performance benefit from scale-out.

VI. Potential Implications

This paper studies the combined impact of node architecture and cloud workloads on network traffic. Based on our experimental results, one can reuse the results to analyze similar

[4]For example, for a 96 dual-core scale-out approach, assuming Mean Time to Failure (MTTF) of three year (26280 hours), the total cost of ownership is $[95 + 26280 \times 0.07 \times 1.7 \times 65/1000] \times 96 = \$28634$ where each dual core machine's CPU and memory cost is \$95 [26] and maximum power usage of each dual core machine is 65 watt [26]. The capital expense (CAPEX) of such configuration is \$9120 and the power cost (≈OPEX) is \$19514.

TABLE VIII: Workload Classification.

Characteristics	**pgrnk**	**fcvt**	**udtc**	**dtc**	**kcore**	**cc**	**apxr**	**dsrv**	**mem**	**CoMD**	**MiniFE**	**HPCCG**
Network bandwidth-intensive	high	medium	high	high	medium	medium	medium	low	low	low	low	high
Network latency-sensitive	low	medium	low	medium	medium	high	medium	medium	medium	low	high	high
Memory-intensive	low	high	low	low	high	medium	medium	high	high	high	high	high
Required bisection bandwidth (Concurrent flow)	medium	medium	high	high	high	high	medium	high	low	high	low	low
Performance/TCO	low (scale-up)	low (scale-out)	medium (scale-up)	low (scale-up)	low (scale-out)	low (scale-up)	medium (scale-up)	medium (scale-out)	medium (scale-out)	high (scale-out)	high (scale-out)	high (scale-out)

TABLE IX: Implications of the metrics of interest.

Metric	graphlab (pagerank, directed triangle count, undirected triangle count)	graphlab (approximate diameter, connected components)	Mantevo (MiniFE, HPCCG, CoMD)	cloudsuite (memcached, data serving) graphlab (kcore)
Comm/Comp overhead	high	high	low	low
Inter-arrival times	lognormal	gev	lognormal	gev
Packet Size Distribution	Bimodal distribution with more large messages	Bimodal distribution with more large messages	Bimodal distribution with more large messages	multi modal distribution with more smaller messages
Bandwidth Requirement	large burst small duration flows	large burst small duration flows	large burst small duration flows	Small burst long duration flows
Self Similarity	high	medium	high	low
Summary	Network bandwidth-intensive with better performance/cost for scale-up	Network latency sensitive with better performance/cost for scale-up	Memory intensive with better performance/cost for scale-out	Stable flows, long duration small packets, with better performance/cost for scale-out

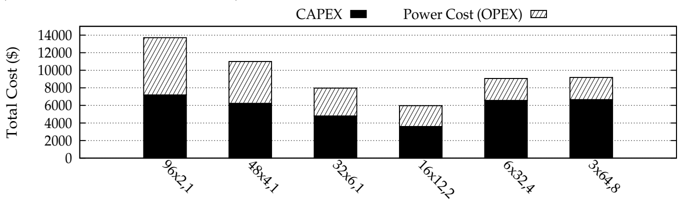


Fig. 9: Total cost of ownership for scale-out and scale-up approaches for various server types (”A×B,C” means A times B-core C-socket server).

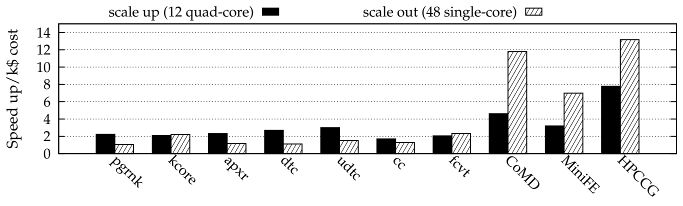
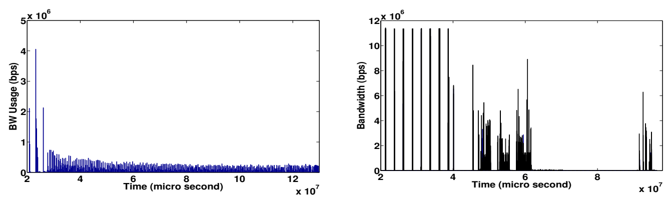


Fig. 10: Speed up per kilo dollar cost for scale-out (48 single core machines) and scale-up (12 quad-core machines).



(a) kcore (b) directed triangle count

Fig. 11: Short burst long duration flow in kcore and high burst short duration flows in directed triangle count.

studied workloads. For example, inter-arrival times and packet size distribution can be used to regenerate the same traffic pattern. In addition, inter-arrival times imply a large amount of self-similarity. Based on our flow analysis, applications with high bandwidth demands and long-lived flows make the network buffer queues to grow faster until the packets are discarded. As a result, TCP congestion control is not fair for applications with low bandwidth demand and short-term flows. Our experimental results opens up the area of designing better congestion control mechanism and buffer managements for different classes of workloads. Based on our observations, we classified the applications based on their bottleneck resources in Table VIII. Based on the results, applications are categorized into five different categories. It is observed that for the same number of cores, scale-out and scale-up approach can have completely different performance/cost metrics. The results can be used for system administrator to get use of heterogeneous designs for different applications. Table IX shows the implication of each metric of interest we discussed on each of the benchmarks we studied. The first class of benchmarks is network-bandwidth intensive benchmarks with better performance/cost for scale-up. The second class of benchmarks is more sensitive to the routing delay. The third class of benchmarks is memory-intensive applications, which get more memory resources in the scale-out approach. The last class of benchmarks have long duration flows with low burst, so these kind of traffics are more stable and are not sensitive to routing delay of high delay switches and can be routed through optical switches. In addition, we discussed how much bandwidth demand is increased when scaling up the server nodes and proposed an analytical model to predict network bandwidth demands of k-core servers. The results of this analysis can be used for network capacity planning. Knowing the characteristics of traffic behavior in advance, can enable one employ a hybrid network architecture, where long lived flows can be routed through optical switches and short duration flows use traditional electrical switches. Based on our experimental study, we can divide the benchmarks in our experimental suite into two groups of long-duration, low-burst flows and short-duration, high-burst traffics. Figure 11 plots two different benchmarks with different flow duration and bandwidth requirement. Figure 11a shows the bandwidth usage of two different server nodes, which shows a long short burst flow, that lives for the whole duration of running the benchmark. Figure 11b shows the aggregated bandwidth usage of all nodes during time, which shows short duration high burst flows. Classifying the network traffic into these groups can potentially help us in exploiting hybrid network architectures. For instance, we can take advantage of both optical switching and packet switching routers.

VII. Related Work

There are recent works aimed at improving the performance of datacenter networks to sustain huge amount of data communication between servers and with outside world. For example [13, 14] suggest to use commodity switches in a fat tree topology for large-scale datacenters. However, managing routing algorithms to utilize the available bandwidth in such networks is very challenging and without a deep understanding of traffic and workload characteristic it is not possible to design a suitable routing algorithm for these kinds of networks.

While there are lots of works to improve the routing algorithms and network topology of datacenters, there is only few works to study the effect of node architecture on traffic pattern of datacenters. Benson et al in [12] have studied traffic behavior of 10 different datacenters using SNMP traces. The packet size distribution of different datacenters in this study follows a bimodal distribution, which is the same as what we found for most Map-Reduce applications. However using SNMP traces it is not possible to poll the switches very often so this study does not look into fine grain traffic behavior of different applications and does not study the effect node architecture on traffic behavior of datacenter networks. Ersoz et al [11] implemented a real 3-tier cluster-based datacenter, and characterize the network traffic behaviour of the nodes. They found that the distribution of inter-arrival times and message sizes of the incoming requests, conform lognormal distribution, and also Pareto distribution is probable for service times of the requests. However, they haven't studied inter-node communications among different servers. Chodnekar et al [44] characterize the network traffic behaviour of the interconnection network of a system when multiple parallel applications are running in the system. They investigate the distribution of message sizes and generation times as a common distribution using SPASM simulator with a dynamic and static strategies.

VIII. Conclusion

In this study, we conducted a workload characterization of wide range of modern cloud applications on a variety of node and network architectures. The results from this study can help us understand cost performance trade-offs in designing datacenters. One of the findings of our simulation-based study is that the inter-arrival times of packets follows a self-similar distribution and increasing the number of nodes tends to increase this self similarity. It is also shown that different benchmarks have different performance/cost for scale-out and scale-up approaches and changing hardware architecture, like CPU core architecture and memory hierarchy of a node can change the traffic pattern on the network and upgrading the servers without changing the network infrastructure, may lead the network to be the bottleneck in the system. We also observed that, some of the benchmarks send a high duration flow into the network, which makes them a suitable fit for hybrid datacenter networks where stable flows can pass through optical switches, whereas low-duration flows can pass through electrical switches.

References

[1] J. Dean et al. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 2008.

[2] K. H. Lee et al. Parallel data processing with mapreduce: a survey. *ACM SIGMOD Record*, 2012.

[3] F. Farhat, D. Z. Tootaghaj, Y. He, A. Sivasubramaniam, M. T. Kandemir, and C. R. Das. Stochastic modeling and optimization of stragglers. *IEEE transaction on Cloud Computing (TCC)*, 2016.

[4] F. Farhat. Stochastic modeling and optimization of stragglers in mapreduce framework. Master's thesis, The Pennsylvania State University, 2015.

[5] F. Farhat, D. Z. Tootaghaj, A. Sivasubramaniam, M. T. Kandemir, and C. R. Das. Modeling and optimization of straggling mappers. Technical report, Technical Report CSE-14-006, Pennsylvania State University, 2014.

[6] S. M Rumble et al. It's time for low latency. In *USENIX, HoOS*, 2011.

[7] M. Alizadeh et al. Data center tcp. In *SIGCOMM*, 2011.

[8] D. G. Andersen et al. Fawn: A fast array of wimpy nodes. In *ACM SIGOPS*, 2009.

[9] V. J. Reddi et al. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *ISCA*, 2010.

[10] S. Kandula et al. The nature of data center traffic: Measurements & analysis. In *SIGCOMM*, 2009.

[11] D. Ersoz et al. Characterizing network traffic in a cluster-based, multi-tier data center. In *In ICDCS*, 2007.

[12] T. Benson et al. Network traffic characteristics of data centers in the wild. In *ACM SIGCOMM*, 2010.

[13] M. Al-Fares et al. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[14] A. Vahdat et al. Scale-out networking in the data center. In *Micro*, 2010.

[15] C. Guo et al. Dcell: A scalable and fault-tolerant network structure for data centers. In *ACM SIGCOMM*, 2008.

[16] R. Appuswamy et al. Scale-up vs scale-out for hadoop: Time to rethink? In *SoCC*, 2013.

[17] `http://parsa.epfl.ch/cloudsuite/`.

[18] P. Lotfi-Kamran et al. Scale-out processors. In *ISCA*, 2012.

[19] M. Ferdman et al. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *SIGPLAN*, 2012.

[20] `http://mantevo.org/`.

[21] `http://graphlab.org/`.

[22] J. Gonzalez et al. Y. Low. Graphlab: A new framework for parallel machine learning. In *CoRR*, 2010.

[23] L. Tang et al. The impact of memory subsystem resource sharing on datacenter applications. In *ISCA*, 2011.

[24] A. Khonsari et al. Mathematical analysis of buffer sizing for network-on-chips under multimedia traffic. In *ICCD*, 2008.

[25] `http://www.amd.com/en-us/products`.

[26] `http://newegg.com/`.

[27] `http://cotson.sourceforge.net/`.

[28] D. Z. Tootaghaj. evaluating cloud workload characteristics. Master's thesis, The Pennsylvania State University, 2015.

[29] `http://developer.amd.com/simnow-simulator/`.

[30] E. Argollo et al. Cotson: Infrastructure for full system simulation. In *ACM SIGOPS*, 2009.

[31] M. Arjomand, A. Jadidi, M. T. Kandemir, A. Sivasubramaniam, and C. Das. Mlc pcm main memory with accelerated read. In *International Symposium on*

Performance Analysis of Systems and Software (ISPASS). IEEE, 2016.

[32] M. Arjomand, A. Jadidi, A. Shafiee, and H. Sarbazi-Azad. A morphable phase change memory architecture considering frequent zero values. In *29th International Conference on Computer Design (ICCD)*. IEEE, 2011.

[33] A. Jadidi, M. Arjomand, and H. Sarbazi-Azad. High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement. In *Proceedings of the 17th IEEE/ACM international symposium on Low power electronics and design*. IEEE Press, 2011.

[34] `http://snap.stanford.edu/data/`.

[35] T. Schank. Algorithmic aspects of triangle-based network analysis. *Phd thesis, University Karlsruhe*, 2007.

[36] U. Kang et al. Hadi: Fast diameter estimation and mining in massive graphs with hadoop. *Carnegie Mellon University, School of Computer Science*, 2008.

[37] M. S. Taqqu et al. Estimators for long-range dependence: An empirical study. *Fractals*, 1995.

[38] W. E. Leland et al. On the self-similar nature of ethernet traffic. *In IEEE/ACM Transactions on Networking*, 1994.

[39] Z. Sahinoglu and S. Tekinay. On multimedia networks: self-similar traffic and network performance. *In IEEE Communications Magazine*, 1999.

[40] K. Park et al. Effect of traffic self-similarity on network performance. In *IPCNS*, 1997.

[41] J. Pješivac-Grbović et al. Performance analysis of mpi collective operations. *Cluster Computing*, 2007.

[42] D. Hardy et al. An analytical framework for estimating tco and exploring data center design space. *ISPASS*, 2013.

[43] C. Kozyrakis et al. Server engineering insights for large-scale online services. In *MICRO*, 2010.

[44] S. Chodnekar et al. Towards a communication characterization methodology for parallel applications. In *HPCA*, 1997.