Benchmarking the Ethernet-Federated Datacenter

M. Gusat, C. DeCusatis, C. Minkenberg,
L. McKenna, K. Bhardwaj
IBM Research, Zurich Research Laboratory
IBM Systems and Technology Group, Poughkeepsie
IBM Tivoli, Australia Development Laboratory
mig@zurich.ibm.com

G. J. Paljak, A. Patariczа, I. Kocsis
Budapest University of Technology
and Economics, Hungary
paljakg@sauron.inf.mit.bme.hu

Abstract— **Currently, datacenters (DC) are the largest closed-loop systems in IT, growing toward multi-million node Clouds. Benchmarking, optimizing and managing such large systems is an active area of research and an open challenge, driven by the trend toward stricter service level agreements (higher performance) and lower power. To evolve the DC design and management from today's ad-hoc solutions to a rigorous discipline, we argue the need for benchmarking based on advanced monitoring and modeling methods. In addition to a positional report of work in progress—our primary target—here we contribute a three-pronged DC benchmarking proposal. Our approach combines resource monitoring and workload characterization with DC performance modeling, both by simulation and by analytical means. While the monitoring and transaction tracing module is still at an early stage, our modeling methods have already been successfully tested in the two study cases. They build on the recent progress made in key independent fields: Ethernet DCB monitoring on L2, ARM-based transaction tracing, large-scale High-Performance Computing (HPC) simulations, and analytical modeling of dynamic distributed systems.**

Keywords - datacenter, performance, benchmarking, 802, Ethernet, network, model, load, transaction, trace, ARM

I. INTRODUCTION

Cloud computing is an emerging method of delivering IT services, in which applications, data, and resources are rapidly provisioned without requiring the end user to purchase or maintain these resources. In a cloud-computing datacenter (DC), applications are provided as standardized offerings to end users over a dynamically reconfigurable network. This enables a more flexible cost and pricing business model, and allows the DC to deploy new resources and technologies much faster than with current methods. This, in turn, is driving significant changes in the way computational resources are used, benchmarked, and optimized.

The traditional DC has taken advantage of the low cost of computing power to proliferate large quantities of under-utilized servers, storage, and networking, often without a clear picture of how the resulting system may perform. As the DC applications and workloads evolve, this approach leads to inefficient use of resources and power. In an effort to address these issues, the modern DC is moving towards new architectures based on hierarchical clustering of virtualized servers around a converged message-passing DC network (DCN). An ensemble of one or more such virtualized DCs is called a compute cloud. Cloud computing uses scale-out IT building blocks to improve the cost/performance ratio.

Regarding the networking infrastructure, currently entailing LAN (Ethernet), StAN (Fibre Channel, FC) and SAN (InfiniBand Architecture, IBA, or Myrinet) solutions, eventually the DC/Cloud traffic will aggregate onto a converged DCN fabric, likely based on the 802 datacenter bridging (DCB) Ethernet running at 10–100 Gbit/s. Although the opportunity exists to consolidate many under-utilized links onto a single, higher-data-rate fabric, the increase in resource utilization must not be allowed to impact user-perceived application performance. Recognizing this, many cloud service level agreements (SLAs) enforce stricter limits on transaction latency, jitter, and sustained throughput; these, in turn, drive requirements for improved scheduling and network congestion control. It remains to be demonstrated whether a converged network can deliver best-of-breed performance if compared with a dedicated, single-purpose network. Furthermore, some new Ethernet-federated DCN fabrics offer significant scale-out capabilities by using virtual chassis backplanes. This has the potential to scale converged fabrics to port counts significantly higher than anything currently deployed. New benchmarking methodologies will be required to design, optimize, and manage these fabrics.

While performance optimization on a component level for individual servers is well developed, it is definitely less mature on an integrated systemic level. Cloud computing is more conducive to system-level analysis than its predecessors, and thus requires a new approach to DC benchmarking. The workloads are heterogeneous, asynchronous, and not well understood at the DCN level; furthermore, cloud-computing DCs incorporate a mixture of mainframes and other types of servers. Traditional component-level benchmarks include the SPEC benchmarks often used for CPU and cache profiling [38], or network component benchmarks, such as [39], often for examining NICs and network protocols. Other efforts such as TPC-W benchmarks for On-Line Transaction Processing (OLTP) [30], although intend to meas-

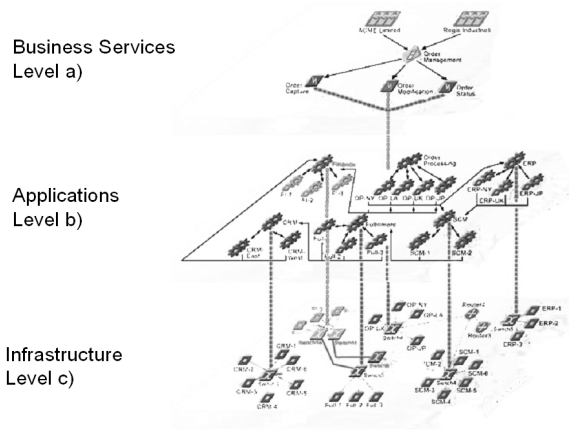

Figure 1. Abstraction levels of a DC structure

ure a complex systems' overall performance, must by necessity often focus on only a few main components: web service engines [34], message-oriented middleware [35], the Java™ Virtual Machine [36] or storage systems [37]. While these approaches remain relevant, they may no longer correlate to the overall cloud DC and DCN system performance. An emerging new class of holistic DC benchmarking that is application- and network-centric must include global knowledge of the current DC workloads and their traffic patterns. Many of these remain unknown today and must be discovered through extensive instrumentation, monitoring, and modeling. Ideally, the resulting benchmarking approach will accurately characterize the DC workload and performance, will scale beyond the 1M-node Cloud and run with low overhead.

In this paper, we study the effect of DC resource configuration on the overall performance. We address the problem of DC-level qualitative and quantitative performance evaluations, in order to both maximize performance—which may be constrained by SLAs—and minimize costs, including energy consumption. These (conflicting) objectives translate into optimizing a distributed system by making it adapt to dynamically changing setpoints and operating conditions, while reducing power and over-provisioning.

The paper is organized as follows: In section II, DC monitoring is introduced. First, in section II.A we deconstruct a typical DC to understand the system structure, the distributed resource topology imposed by the interconnection network graph, and finally the workload mapping. Next, in section II.B we investigate the challenge of projecting business transactions to infrastructure-level flows, focusing on transaction tracing. This section is intended as a tutorial and field review of the most recent related work. The monitoring part combines transaction tracing with analytics to correlate events, extract deterministic or statistical transaction features, and generate equivalent traffic for use in modeling. In section III we describe our two established modeling methods for DC/Cloud performance evaluation, i.e. simulation and analytics. Each method is exemplified with a case study showing recent results of our proposed method. We conclude in section IV, also including a discussion of future work.

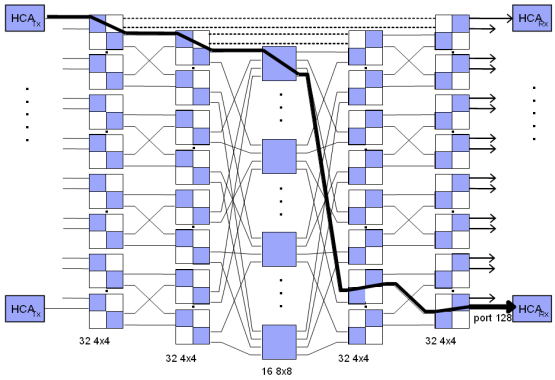

Figure 2. A typical graph of the infrastructure level: DCN fat-tree topology. Bold: one of the multiple available routing paths e2e

II. DATACENTER MONITORING

A. *Deconstructing the DC: A Topological View*

We examine the effect of the DC structural configuration on the overall performance. To describe the structural configuration, we introduce a three-pronged framework, depicted in Figure 1. It is expressed on three levels of abstractions by three graphs: (a) The nodes on the business topology level are services, and edges connect two nodes if one service uses or relies on the other. (b) The application level contains all deployed software components as nodes and their interdependencies as edges. (c) The infrastructure level is composed of computational and network device nodes; the edges of this level are the network links. Mapping between the three graphs exists: in the computing cloud services are mapped onto their provider applications, and applications onto infrastructure resources used. To understand a system, each level and the mapping between the levels must be understood.

B. *Transaction Tracing in DC. Related work.*

1) *Transactions in datacenters*

A transaction is a causal data flow from the entry point of a request to the exit point of the response, initiated by an actor of the system. A transaction represents a trajectory in the DC state space. Transactions on each abstraction level of the DC topology (Figure 1) appear as ordered paths on the graph. On level *a*, a transaction is a service invocation, which may also rely on other services to serve the response. Inside the service provider, on the application level (level *b*); a transaction is a series of method invocations, generally spanning multiple applications with RPC-style interactions. Finally, a transaction translates into an ordered set of resource oc-

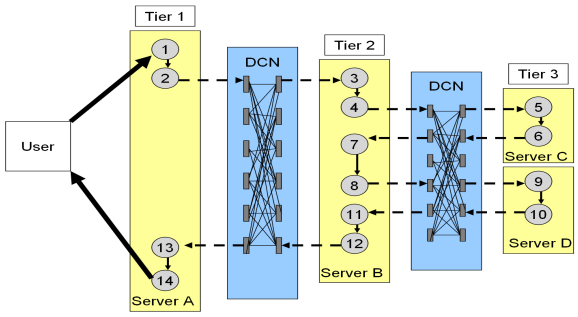

Figure 3. The ARM projection of a transaction on the DC infrastructure

cupancies: a path of network packet flows and computation times on the infrastructure level, flowing through multiple servers and the interconnecting DCN of the infrastructure topology (level *c* and Figure 2).

2) Transaction-Tracing Methods. Overview

The task of monitoring, in this case, is to track transactions following the causal path of component-level resource occupancies. This provides performance information broken down into components and traces that can later be used for driving the simulation or validating models.

There are three main types of transaction observation and reconstruction methodologies: white-, gray-, and black-box. In a *white-box model*, all source code is available and can be freely modified, instrumented; either application-specific assumptions or globally unique identifiers (GUIDs) are used in most cases. This is the case for NetLogger [9] or the Application Response Measurement standard (ARM, [29]) where the application source code is extended with explicit tracing information. WebMon [28] uses cookies modified for storing GUIDs that are created by custom JavaScript, and these are passed to instrumented web and application servers. User Programmable Virtualized Networks, described in [33], provide the developer with the freedom to handle network interactions (including tagging packets with ID), as well as to extend the application and to modify the typical OS kernels. However, in general the application source code is not available, or it is problematic to modify owing to its complexity or other reasons. In this case, platform-level instrumentation may still be feasible, which involves the extension of OS components, e.g., protocol implementations, middleware applications or runtime environments, to support tracing.

Next, we look at some *gray-box solutions*, Magpie [10] is built on Microsoft® Windows® platform and uses a built-in event-logging framework for reconstructing causal paths, whereas for distinguishing between threads, auxiliary events and application-specific schemas are used (please note that earlier versions of Magpie used GUIDs). PinPoint [11] uses platform-specific tracers that extend components, such as the web server and several J2EE containers, to tag transactions with GUIDs and to preserve the tagging. A similar instrumentation

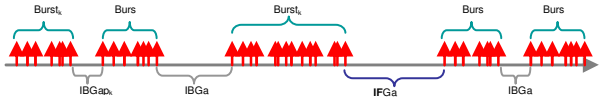

Figure 4. Transaction consisting of 5 bursts, each having 7-15 Ethernet frames, with an arbitrary burst size and inter-burst gap.

for the Java Virtual Machine [12] uses agents in the runtime-environment and inserts trace statements ahead of the flow control. CORBA interceptors are also used, together with the ARM standard [31] or focusing on statistical approaches [21]. For CORBA/COM, a method is introduced that uses automatically generated skeletons and stubs to handle GUIDs [27]. In [13], ISA level instrumentation is used, i.e., a tag mapping is maintained for every byte of the memory and tags are inserted into and extracted from Ethernet frames. Whodunit [26] detects transactions communicating through shared memory, events or RPCs by means of an extensive, but low-overhead instrumentation. SysProf [25] focuses on profiling transactions, keeping track of resource utilization using kernel-level instrumentation, but requires additional supplied knowledge to identify interleaving transactions.

At the other extreme, the most generic case is the *black-box approach*, where no previous knowledge on the components is provided, and only passive monitoring instruments (with practically zero performance impact) are used. The causal-path reconstruction is based on probability and statistics. In their mathematical overview of the problem [15, 16], the authors present two algorithms: harnessing nested sub-transactions (identifying call-pairs and matching the probably nested tuples), and a convolution method applied on message traces as time signals. E2Eprof [14] analyses log files and estimates most probable causal paths based on cross correlation. In [17] and [20]. a network-only approach focusing on TCP is introduced; and in [23] the mass characteristics of transactions are analyzed.

3) Our Approach: ARM/ITCAM

We use a grey-box scenario on the concept of extending middleware components. We take advantage of middleware applications offering the ability to plug in additional code modules by extracting attributes about a transaction and sending that information to a separate tracking processor. One example of such transaction tracking is the IBM® Tivoli® Composite Application Manager (ITCAM) for Transactions; this software is based on automatic insertion of ARM calls to the control flow and harnessing native instrumentation of supported middleware. Transaction tracking uses the techniques of linking (a single attribute is used to group events) and stitching (several attributes are combined using a predefined method) to correlate transactions end-to-end (Figure 3).

The challenge of transaction tracing for further analysis and trace-driven simulation is two-fold:

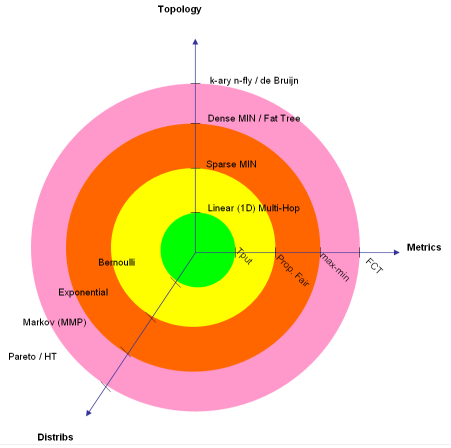

Figure 5. DCN Benchmarking axes. Topology is decided by the DC architect and the DCN vendor; Metrics are imposed by the SLAs; Traffic Distributions depend on the DC workload characteristics.

(i) The lack of a de-facto standard DC communication protocol similar to the Message Passing Interface (MPI), typically used for instrumentation tap, in HPC. In DCs, instead of MPI calls, there are RPC, CORBA, JDBC, etc. calls, to name just a few. There is a multitude of protocols, some of which are proprietary and so may be their implementation. Even when one has an instrumented version of one or some of these protocols, generalization is an open issue.

(ii) Observing and rebuilding causal paths (trajectories) are demanding in a typical DC environment. Transactions span across multiple, different-purpose subsystems and protocols; most of the information exchange is encrypted which is, by design, against observability.

4) DCN-level Benchmarking Workload and Metrics

The application transactions mapped to the network layer translate into flits, packets/frames, and flows. A given *workload* entails the following components, listed hierarchically: work, job/transaction, flow, burst, and packet (frame, flit). Each component is described by two random variables (e.g. burst size and inter-burst gap), for which we must choose a distribution based on the workload characteristics of the specific benchmark (inter-burst/frame gap, Figure 4).

The best established performance *metrics* at DCN level are latency (end-to-end (e2e) delay at L7), throughput, jitter, and, at times, fairness. More recently [6], argued for the introduction of Flow Completion Time (FCT) as more representative of the user experience. FCT is the time interval between the injection of the first Ethernet frame by the source node and the reception of the last frame at the destination node. One problem on L2, where DCB is defined in 802, is the flow definition – which differs from that of a L3/4 (TCP) flow. Hence, the two main challenges of FCT as a DCN metric are as follows:

1) Sensitivity to distributions renders the choice of distribution a delicate issue. For example, for Pareto distributions, FCT loses its relevancy because

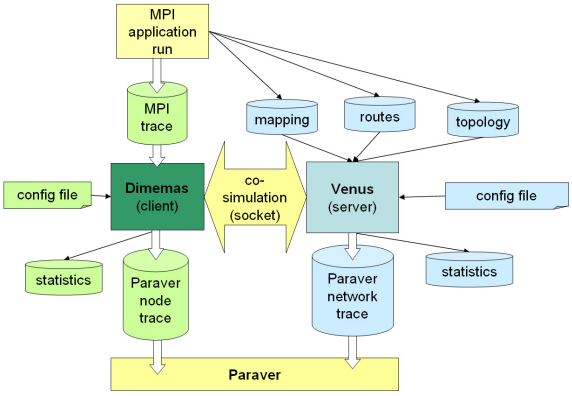

Figure 6. VENUS simulation platform.

$FCT = \Sigma (t_{inject} + t_{queue} + t_{flight} + t_{RTX}) \neq L_{e2e}(X)$, i.e., the central limit theorem does not apply. Therefore, each term of the sum (except for t_{flight}) must be analyzed and reported independently.

2) The presence of priority flow control (PFC) as defined in and required by most DC applications, including Fiber Channel over Ethernet (FCoE), requires *precise* definition of flow completion and rigorous accounting in the simulation statistics of: (1) flows received entirely without any loss; (2) flows received entirely with some loss; (3) flows received partially, and (4) flows not yet having arrived at destination.

As none of these above issues has been practically solved (for a solution see [1]), although FCT has been proposed as a DCN benchmarking metric, it was not pursued in 802 DCB. Hence, the metrics used most in DCs are latency and throughput (primary), and power, fairness and jitter (secondary). Although power is expected to become a primary metric, we currently cannot properly monitor and aggregate all power statistics into a meaningful metric, such as [TPS/Watt]. Another open issue is how to homogenize the L7/application metrics—e.g. TPS and response time—with the L2 DCN metrics, where throughput and latency represent aggregate statistics. The translation between application and DCN (L7:L2) metrics is the role of "integrated" metrics conversion—a problem yet to be solved.

The DCN-specific benchmarks used here can be based either on traces or on synthetic traffic generators. For the former case, the trace format includes {(1) TimeStamp | (2) SRC | (3) DST | (4) Prio | (5) BSize}; in addition, field (1) is further extended with the GlobalID from transaction-tracking. The latter case of synthetic generators is more elaborate [1], evolving along three axes as in Figure 5.

III. DATACENTER MODELING

A. Simulation Modeling

Our approach for modeling is based on structural information (i.e. topology graphs) about the system and

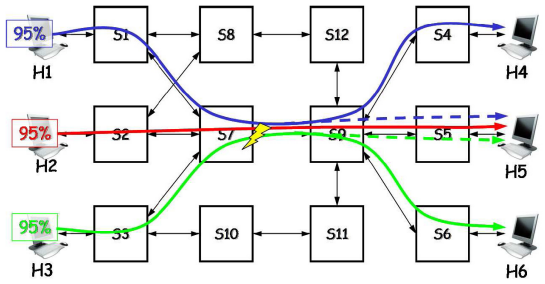

Figure 7. Simulated DCN topology.

the concept of transactions being ordered paths on topology graphs. The main requirements of a suitable simulation environment are architectural accuracy (all relevant DC details being captured), scalability to full DCN size. and execution speed. We draw from our experiences in modeling HPC systems, where the two main simulation methods are "Monte Carlo" and trace-driven.

In a "Monte Carlo" type simulation synthetically generated workloads, based on predetermined probabilistic distributions, drive accurate functional models. In a trace-driven simulation, computing nodes are represented by a trace that contains two basic kinds of records, namely computation and communication, rather than by an exact model of their behavior. Computations are not actually performed, but represented by the amount of CPU time they would consume in reality. Communications are transformed into data messages that are fed to a model of the DCN. To ensure accurate results, the simulation should preserve causal dependencies between records, e.g., when a particular computation depends on data to be delivered by a preceding communication, the start of the computation must wait for the communication to complete.

As many HPC applications are based on MPI, tracing MPI calls is a suitable method to characterize the communication patterns of HPC workloads.

An example of this approach is the MARS simulator presented in [18]. Here we give a brief overview of its successor, the VENUS-Dimemas HPC tracing and simulation environment, depicted in Figure 6. This is a co-simulation environment in which Venus is responsible for detailed simulation of the network and Dimemas for replaying application traces, i.e., simulating computation nodes. Paraver processes the simulation output and provides a graphical representation of the state of MPI threads and of the communication between them and network devices. The state of an MPI thread is an activity (idle, running, waiting, etc.) and the buffer-filling level of the network level. The inputs for the simulation environment are the following: (a) network topology descriptor; (b) a route descriptor for explicit definitions of routes between any two hosts; (c) network device models (representing Myrinet DCN hardware in this case),

Parameter	Value	Unit	Parameter	Value	Unit
line rate	10	Gb/s	mean sample itvl.	150	KB
frame size	1500	B	buffer size per port	150	KB
min. rate	100	Kb/s	fast recovery thr.	5	
Q_{eq}	33000	B	byte count limit	150	KB
W_d	2		active incr.	5	Mb/s
G_d	1/128		hyperactive incr.	50	Mb/s
quantization	6	bit	min. decr. factor	0.5	
timer	15	ms	extra fast recovery	enabled	
AR timer	250	ms			

Figure 8. VENUS Simulation parameters for QCN-based AR.

and (d) MPI application run traces and task mappings. For further details, the reader is referred to [8].

The goal of the simulation is to analyze the system and to evaluate what-if scenarios, i.e., to learn about the interoperation of components. These are important when designing for a given performance metric (whether it is defined by an SLA or is a dynamic efficiency objective).

Our proposal is to use the verified and validated MPI environment, and to replace the original MPI traces by commercial DC traces. According to our approach, business transactions in a cloud would translate into a causally ordered set of communication and computation primitives. Proper instrumentation is the only way to be able to trace transactions in a heterogeneous DC environment, as shown in section II, and this can be extended as far as providing transaction information on L2 for wide usability and low overhead.

*1) **Case Study I**: 802.1Qau-based Adaptive Routing*

The recent LAN-SAN-StAN consolidation efforts and the emergence of competitive IBA products have increased the demand for a DC version of Ethernet with multiple priority classes to support storage, clustering, and LAN traffic. The sensitivity of DC applications to latency and frame loss disfavor heavy protocol stacks, or, in large multihop fabrics, the e2e re-transmissions as in TCP. Several working groups in the IEEE and IETF Standards bodies are addressing key issues to ensure that 10GE meets DC and HPC requirements. In IEEE, the *Data Center Bridging* (DCB) Task Group is responsible for defining the 802 Standards for congestion management (802.1Qau), traffic differentiation (802.1Qbb), and enhanced transmission selection (802.1Qaz). Similar to HPC and IB networks, DCN applications favor fast L2 implementations in hardware. The DCB-Ethernet choice of 8 hardware priorities, each independently flow-controlled (as IB's 15 Virtual Lanes), however, was argued at great lengths in 802—until an acceptable trade-off could eventually be reached, namely, a Priority Flow Control (PFC, defined in 802.1Qbb, also formerly known as Per-Prio-Pause). This scheme required by the storage and the other DC applications will be IEEE-standardized *in conjunction* with a (distinct) congestion management (CM) scheme that can be clearly specified and contained to the DCN domain. This CM scheme, also known as QCN [2,4,24,39], was defined in

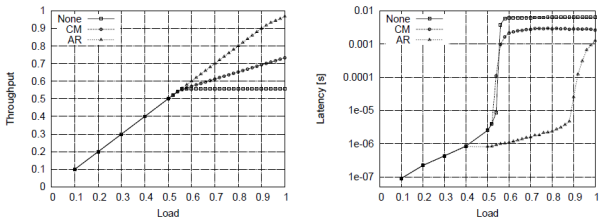

Figure 9. Uniform traffic results: w/o CM, w/ CM, w/ CM+AR.

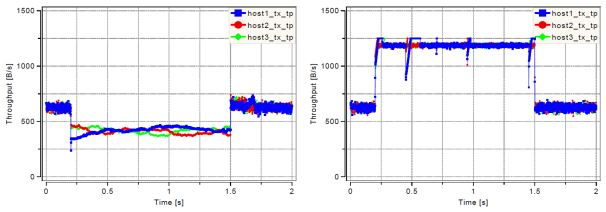

Figure 10. Congesting traffic T_{put} results: left: QCN; right: AR.

802.1Qau to prevent hotspot congestion spreading in saturation trees [5,7], to stabilize the network operation under heavy load, and to improve the overall DCN performance.

While both DCB schemes, PFC and QCN, are now being drafted to be approved as 802 standards, the debate on QCN's merits is still ongoing. Some DC vendors and customers question QCN's scalability, stability, parameter tuning, and its open-loop (load agnostic) recovery—or, in general, QCN's sufficiency for the next decade's DCs and compute clouds. At the other extreme, various experts propose even further simplifications, such as the removal of CN tags. Finally, others argue that for certain applications such as storage and HPC, 802's PFC is sufficient, while CM is either unnecessary or potentially detrimental; instead, improved routing and load balancing would be preferable for DC and HPC.

Therefore, we propose a practical solution to the CM debate: A management scheme that uses the QCN load sensor (i) to re-balance the offered load by adaptively routing (AR) the traffic around hotspots; (ii) *if* all admissible multi-pathing options have been exhausted *and* if the hotspot persists, to enable QCN to perform source-injection rate control. Hence our two-tier approach: In response to congestion detection by QCN, an attempt is made to reroute "hot" flows ("culprits") onto an alternative, uncongested path, and only if no uncongested alternative exists are the culprits' injection rates source-controlled. We thus build on QCN—whose load sensor is already built into DCB-compliant switches—to exploit an inherent property of DCNs: multi-pathing, i.e., the presence of multiple alternative paths between any pair of DC nodes, as is also the case in the fat-tree topology shown in Figure 2. We demonstrate how this can lead to significant performance improvements by taking full advantage of path diversity. Moreover, we highlight the scheme's practical usefulness by showing how it improves the performance of a parallel benchmark program on a realistic network.

802 CM Background: QCN fights saturation trees by pushing the backlog to the edge of the network. By means of congestion-notification (CN) frames, switches instruct the sources of flows contributing to a specific congestion point (*hotspot*) to adjust their send rate to match the aggregate rate of all flows contributing to the bottleneck bandwidth. Reducing the send rates is unavoidable if the congestion is due to *endpoint* contention, i.e., multiple sources sending to the same destination node at a higher aggregate rate than that node can service.

However, congestion can also occur within the network because of *routing conflicts*, i.e., because multiple flows, with possibly diverse destinations, are mapped to a common switch port by the routing algorithm. This could be effectively resolved by rerouting the congested flows onto an alternative path, thereby obtaining higher throughput than QCN, which merely reduces the source rates. *Combining* adaptive routing (AR) with CM can notably decrease the delay and increase the throughput of a congested network, because rates have to be reduced only if no alternative uncongested path exists. Thus, AR provides a *spatial* alternative to the *temporal* reaction of QCN. The benefit is that no changes are necessary to the Ethernet frame format, existing DCB schemes, and Ethernet adapters. AR operates by modifying the routing behavior of the Ethernet switches. Interoperation with 802.1Qau is seamless: if no alternative path exists, the send rates will eventually be reduced to match the bottleneck rate.

VENUS Simulation Results: The aim of our *fully* AR scheme is to exploit the congestion information conveyed by notifications generated by 802.1Qau-enabled switches. These CNs travel upstream from the congestion point to the originating sources of the hot flows. While relaying CNs, the upstream switches *snoop* their content to learn about downstream congestion. By marking ports as congested with respect to specific destinations, a switch can reorder its routing preference of eligible output ports for the destinations affected to favor uncongested ports over congested ones. How the routing is configured to avoid loops, broadcast storms, and adapt to changing loads is described in [40]. We select results of simulating the DCN topology shown in Figure 7 in VENUS, with QCN parameters set according to Figure 8.

The uniform traffic results in Figure 9 show that without CM, throughput saturates at about 55.6% because there were nine hot flows in each direction (the flows from hosts H1–H3 to H4–H6 and vice versa) contending for a bottleneck link, hence each flow received 1/9 of the link rate. As CM was disabled, the input buffers of the congested switches filled up rapidly, causing PAUSE to be applied. As each of the cold flows shared

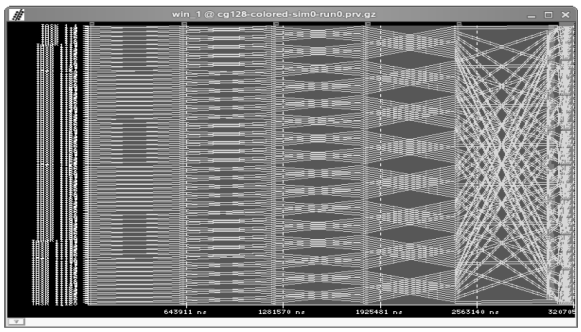

Figure 11. 3.2-ms Paraver trace of optimal CG execution.

a link with at least one hot flow (e.g., *all* flows from hosts H1–H3 traversed a congested link to switch S7, and *all* flows from hosts H4–H6 traversed a congested link to switch S9), the cold flows effectively obtained the same throughput as the hot ones, leading to an aggregate throughput per host of 5×1/9 = 55.6%. Enabling QCN increased the throughput. The CM mechanism controlled the congested queue lengths such that the links leading to the congested switch did not have to be paused continuously. Thus, unlike the case without CM, here the cold flows ("victims") were unaffected. Finally, enabling AR increased the saturation throughput to 96.8% (>99.5% was achieved in other scenarios), indicating that the AR scheme optimally exploited the available path diversity. The mean latency is reduced more than 1000fold at loads between 55% and 85%. Infrequent misorderings occurred, with negligible effect on latency: The mean length of the resequencing buffers was less than one frame. Similarly, enabling AR for non-uniform congesting traffic, i.e., flows contending for resources, nearly triples the throughput per flow (to 1187.5 MB/s, the full offered load) with respect to QCN, which sometimes is also 5:1 unfair.

Application Impact: HPC Benchmarking of AR. To test AR's practical benefits, we selected the Conjugate Gradient (CG) application from the NAS Parallel Benchmarks (NPB) as a suitable candidate because of its high communication demands. To reflect accurately the application behavior, we adopted a tracing method as in Sect. II, but using HPC and MPI: First, the 5-phase CG application was run on a *real* 128-node HPC machine. During the HPC run, all MPI calls were recorded in a trace. The communication pattern is shown in Figure 11.

Then, the trace file was replayed on VENUS. In Figure 12, we observe that when QCN is enabled without AR (d-*mod*-m), the execution times are worse than *without CM*, because the congestion induced in the final phase leads to transmission rate reductions without any hope of improvement, because the problem here is not throughput loss due to saturation tree congestion. Rate reduction can only improve the aggregate throughput if there are "victims" and "culprits," whereas here every flow is a culprit—the definition of which has also been argued in DCB. In addition, the runtime is too short for

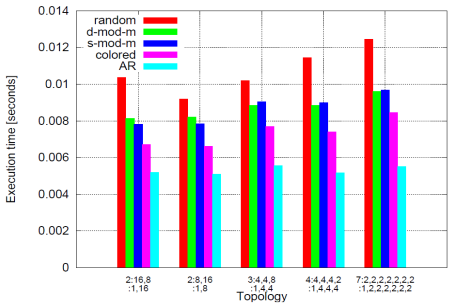

Figure 12. Performance under random task placement: AR outperforms even the offline route optimization ('colored').

QCN, as its rate recovery duration is in the range of 10s-100s of ms, typically > 70 ms. Such results validate the concerns that CM can be detrimental to real applications, and call for rigorous benchmarking before deploying CM in DC and HPC applications. However, large parts of QCN are useful for load monitoring and dynamic load balancing, as proved here by our fully AR proposal.

2) Simulation Challenge: Large Exploration Space

Simulation provides accurate results proportional to how detailed the model has been captured, and to the relevancy of the benchmark, i.e., workload or traffic pattern. This requires the performance modeler to have a deep understanding both of the DC architecture, down to its most intimate details, and of workload characteristics. Simulations, however, also have constraints. For example, in a "Monte Carlo" type simulation of a DC network, the following dimensions are to be explored (partially or exhaustively): (d1) number of nodes; (d2) switch/adapter architecture (buffering, queuing, scheduling, and switch allocation are distinct subdimensions); (d3) topology; (d4) LL-FC settings; (d5) QCN parameters, if enabled; (d6) load balancing and adaptive routing, if enabled; (d7) traffic scenario; (d8) metrics of interest; and (d9) number of simulation points to achieve the prescribed confidence interval. The list includes only the L2-specific dimensions; a complete datacenter model up to L7 has orders of magnitude more dimensions. To achieve correct and feasible simulations or to be able to create closed control loops, first the dimension of the problem must be reduced. This can be done by focused benchmarks with well-defined metrics and by automated methods for data analysis.

B. *Analytical Modeling*

The simulation challenges above can be alleviated by analytical means, e.g. dimension reduction, as will be shown in the ensuing case study. Another use of analytics is in post-processing the traces acquired in Sect. II. These usually must be de-noised, correlated, aggregated, and scaled in time and space. Finally, stability analysis and dynamic response investigations mandate an analytical formulation of the DC system. In an online closed-loop management, the DC/cloud would take an input of a desired value of a performance metric (set

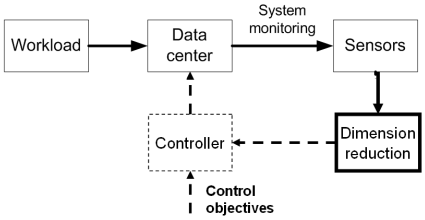

Figure 14. Dimension reduction in a control loop.

point) and optimize its services, i.e., reconfigure itself, grow or shrink the set of resources used.

This is analogous to a control loop, which aims at observing and regulating the dynamic behavior of a plant. In this case, the controlled plant is the IT infrastructure and its exogenous input is the workload.

The sensor on the plant is system management, and the controller is an intelligent decision-making unit capable of provisioning the system to meet or approximate optimization goals (set points) in steady-state operational points. We postulate a standard continuous state space formulation of the DCN dynamic model:

$$\dot{z}(t) = f(z(t), u(t)) \tag{1}$$

$$\tau(t) = g(z(t), u(t)), \tag{2}$$

where $z(t) = [Q(t), \Lambda(t)]^T$ is the internal state vector with initial condition $z(0) = z_0$; it describes the evolution of the internal queues $Q(t) = [q_1(t), q_2(t), \ldots, q_M(t)]^T$ and the injection trajectories, assuming continuous (instead of switched) increase/decrease rates $\dot{\Lambda}(t) = [\dot{\lambda}_1(t), \dot{\lambda}_2(t), \ldots, \dot{\lambda}_N(t)]^T$. $u(t)$ is the exogenous input of $\ell(t) \le L$ links with service rate $\mu_l(t) \le C_{l\max}$ (limited by the per-lane QoS allocation) and $n(t) \le N$ active nodes with injection (driven by the external workload) demand $\lambda_i(t) \le \Lambda_{\max}$. $f(.)$ describes the DCN dynamics, including the forward R^f and reverse R^r routing matrices, AQM method, scheduling, and flow control. $g(.)$ describes the total delay $\tau_{\text{tot}}(t)$, which includes the forward $\tau^f(t)$ (measured delay) and the reverse delay $\tau^r(t)$ (passive delay of receiving feedback in closed loop). The above system can be re-formulated as an H^∞ control with receding horizon prediction (for slow variations of $\dot{\tau}(t)$), in which the source-controlled system with input $u(t)$, state $x(t)$, disturbance $w(t)$, and output $y(t)$ has the form

$$\dot{x}(t) = A(t)x(t) + B(t)u(t - \tau^f(t)) + D(t)w(t) \tag{3}$$

$$y(t) = C(t)x(t - \tau_{tot}(t)). \tag{4}$$

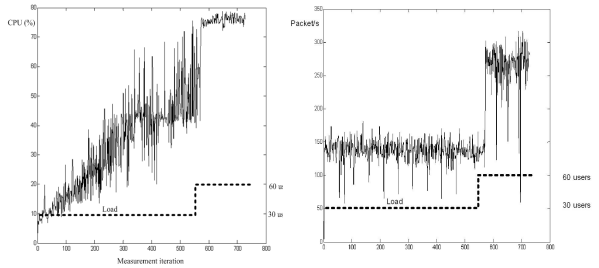

Figure 15. (left) Overall CPU utilization of the Apache HTTP server (%). (right) Number of network packages sent by the database cluster controller (packet/s):Two system monitoring metrics selected as most importand by the mRMR algorithm

Typically, predictive methods mandate that sources declare their future injection rate demands $\Lambda(t) = [\lambda_1(t), \lambda_2(t), \ldots, \lambda_N(t)]^T$ up to the $\tau_{\max}$ horizon ahead. Practically either we need either to relax or, ideally, to remove this strict constraint which is not tenable in Cloud and DCN applications. In either case, we must characterize the maximal horizon based on delay observation statistics.

Next, we focus on analyzing the internal structure of the system and present a system-level modeling approach and model evaluation techniques. We also present a case study for building intelligent sensors to reduce the immense dimensionality of DC performance modeling.

C. *Case study II: Dimension Reduction*

A datacenter is a heterogeneous interoperation of many stand-alone systems with numerous internal tuning parameters. Industry standard components, such as operating systems and middleware applications, offer numerous performance attributes to be measured by monitoring solutions (custom applications or commercial products). Such software provide exhaustive insight into the internal parametric changes of the datacenter, but easily overwhelm system engineers with superfluous data, possibly even hiding important information. It is essential to set up a model of the system that describes it well, i.e., that incorporates all relevant metrics, but removes the complexity of handling vast amounts of performance attribute data.

In [19], the authors present an automated means of identifying a set of highly relevant system-monitoring sensor attributes with respect to an objective function. A feature-selection algorithm for is used the discovery of a subset (of sensor observations) that is relevant to a system-level metric or service-level objective. The relevancy and redundancy are determined by using the mutual information of time series. The goal is to select a subset S of all metrics monitored that meet two requirements: (i) the mutual information of the elements of S against a system-level metric (relevancy) is maxi-

mal, and (ii) the mutual information among the elements of S (redundancy) is minimal. Formally, the mutual information I of probability variables x and y is defined as:

$$I(x;y) = \iint p(x,y)\log\frac{p(x,y)}{p(x)p(y)}dxdy \qquad (5)$$

These criteria can be optimized by iteratively adding elements x_i (representing time series) to S based on the following criterion ($m = |S|$ and S_{m-1} being the set of metrics selected prior to the current iteration):

$$\max_{x_j \in X - S_{m-1}} \left[I(x_j;c) - \frac{1}{m-1}\sum_{x_i \in S_{m-1}} I(x_j;x_i) \right] \qquad (6)$$

The solver algorithm, "minimum-Redundancy-Maximum-Relevancy" (mRMR) was introduced in [22] with a detailed description and case studies.

Our aim is to reduce the dimension of the DC-monitoring data by algorithmic analysis. Here is a brief summary of the key steps: (i) Instrumenting the infrastructure with measurement agents; (ii) Running benchmarks or other workloads on the system; (iii) Collecting internal performance attribute values and values of high-level-integrated metrics as time series; (iv) Selecting a metric as objective function; (v) Finding a minimum redundant set of time series (dimensions) that are maximally relevant for the objective function by iterative solution of (6); (vi) Supplying DC control with this set.

The mRMR algorithm is used in and evaluated for a testbed of a small multi-tier infrastructure with centralized system monitoring. The overall throughput ([transactions per second]) is selected as an objective metric, the load is doubled at 2/3 of the experiment interval (dashed line in Figure 14). Based on the results, we see the need for (a) run-time interaction, for example, after the appearance of server CPU usage in S (Figure 14, right); new servers can be added to that tier to handle the load, and (b) design-time guidance, for example, the number of network packets transmitted (Figure 14, left) being in S can indicate underplanned network link capacity. These are two of the 10 selected metrics of the example result shown in [19].

This approach helps to understand the impact chains inside the system and to identify control attributes. The next step towards realizing intelligent actuation would be to provide a classification of attributes to controllable parameters and a state-machine-like set of rules that is able determine the type of actuation to be carried out.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we have shown the challenges of benchmarking a DCB Ethernet-federated datacenter, and some of the potential methods to overcome these challenges. Moreover, we have confirmed the benefits of rigorous DC benchmarking based on advanced monitoring and modeling. As our main contribution, we have proposed a three-pronged approach to DC management, which combines workload and resource *monitoring* with performance *modeling*, both by simulation and analysis.

These methods build on our recent progress made in key independent fields: Ethernet DCB monitoring on L2, ARM-based transaction tracing, large-scale HPC simulations in VENUS, and analytical modeling of dynamic and distributed systems. Using VENUS to benchmark 802-based AR, we have demonstrated the load-balancing potential of the upcoming 802.1Qau standard. Our first benchmarking results showed that the DCN performance can be *radically* improved, sometimes by orders of magnitude. For example, the fully adaptive routing scheme proposed in Sect III.1 reduces the execution time of the CG benchmark without incurring any of the detriments exhibited by QCN in this case. Next, we have used an analytical sensor selection method—capable of *automatically* processing the DC-monitoring streams—to identify the set of primary metrics that dominate the overall DC operation. Some of these benchmarks and simulation tools have already been used to shape RapidIO, IBA CM, and the upcoming IEEE 802 DCB Ethernet.

Next Steps: With the progress toward cloud computing and mega-node datacenters, system-level optimization and management are areas of growing relevance in IT. The community is making steady progress in DC benchmarking, improved scheduling and load balancing, and ultimately, toward automated Cloud management and optimization. However, a candid assessment of the current situation reveals areas where more effort is needed. Particularly vexing is the lack of an established set of DC benchmarks, or at least, of a guiding methodology to this end. Although most of the standards bodies, researchers, vendors, and customers are asking for DC traces, workloads and traffic generators, no such data is publicly available yet. This we attribute to the following: (i) lack of a standard DC *message-passing library*, similar to MPI in HPC, and (ii) lack of *monitoring instrumentation* capable of (DC and cloud) system-level benchmarking. The latter lack is sustained by the continuing prevalence of segregated, component-level benchmarking: CPU, server, JVM, database. etc. Despite these challenges, DC workload characterization and performance benchmarking constitute a promising area of research for the systems community.

V. ACKNOWLEDGEMENTS

We thank R. Luijten, W. Denzel, G. Rodriguez, Z. Egel, D. Toth, A. Scicchitano, and Charlotte Bolliger for their contributions.

REFERENCES

[1] Gusat, M., et al., "On Flow Completion Time Benchmarking in Datacenters," http://www.ieee802.org/1/files/public/docs2007/au-sim-ZRL-FCT-BMRK-r03.pdf , 2007

[2] Pan, B. Prabhakar, and A. Laxmikantha, "QCN: Quantized Congestion Notification," 2007, www.ieee802.org/1/files/public/docs2007/au-prabhakarqcn-description.pdf

[3] Bergamasco, "Ethernet Congestion Manager (ECM) Specification," Cisco Systems, Draft EDCS-574018, Feb. 2007.

[4] "IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks - Amendment: 10: CN," http://www.ieee802.org/1/pages/802.1au.html, 802.1Qau.

[5] Pfister et al., "Solving hot spot contention using InfiniBand Architecture congestion control," *Proc. HP-IPC 2005*, Research Triangle Park, NC, July 2005.

[6] Dukkipati, McKeown. "Why flow-completion time is the right metric for congestion control," *SIGCOMM Comput. Commun. Rev.* 36, 59-62, 2006

[7] Pfister, Norton, "Hot spot contention and combining in multistage interconnection networks," *IEEE Trans. Computers*, vol. C- 34, no. 10, pp. 933–938,. 1985.

[8] Minkenberg, C., Rodriguez, G., "Trace-driven co-simulation of high-performance computing systems using OMNeT++," *Proc. 2nd Int'l Workshop on OMNeT++*, 2009

[9] Gunter et al., "Log summarization and anomaly detection for troubleshooting distributed systems," *Proc. 8th IEEE/ACM Int'l Conf. on Grid Computing*, 2007

[10] Barham et al., "Using Magpie for request extraction and workload modelling," *Proc. 6th USENIX OSDI*, 2004.

[11] Chen et al., "Path-based failure and evolution management," *Proc. 2004 USENIX Symposium on Network Systems Design and Implementation,* 2004.

[12] Mirgorodskiy, Miller, "Diagnosing distributed systems with self-propelled instrumentation," Technical Report, University of Wisconsin,, 2007

[13] Mysore et al., "Understanding and visualizing full systems with data flow tomography," *Proc. ASPLOS 2008*.

[14] Agarwala et al., "E2eprof: Automated end-to-end performance management for enterprise systems," *Proc. DSN 2007*.

[15] Aguilera et al., "Performance debugging for distributed systems of black boxes," *Proc. ACM Symposium on Operating Systems Principles (SOSP) 2003.*

[16] Anandkumar et al., "Tracking in a spaghetti bowl: monitoring transactions using footprints," *Proc. ACM SIGMETRICS,* 2008

[17] Liu et al., "Real-time Application Monitoring and Diagnosis for Service Hosting Platforms of Black Boxes", *Proc. 10th Symposium on Integrated Network Management*, 2007

[18] Denzel W., et al., "A framework for end-to-end simulation of high-performance computing systems." *Proc. 1st Int'l Conf. on Simulation Tools and Techniques for Communications, Networks and Systems*, 2008, article 21.

[19] Paljak et al., "Sensor Selection for IT Infrastructure Monitoring," unpublished.

[20] Ozturk, LaFon, "DAFA: Distributed Application Flow Analyzer," *Proc. Fifth Int'l Conf. on Information, Communications and Signal Processing*, 2005, pp. 404-408, 2005

[21] Moe et al., "Using Execution trace data to improve distributed systems," *Proc. Int'l Conf, on Software Maintenance (ICSM'02)*, 2002.

[22] Peng et al., "Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226-1238, 2005.

[23] Chen Yoshihira. "Modeling and tracking of transaction flow dynamics for fault detection in complex systems," *IEEE Trans. Dependable Secur. Comput,* vol. ?, pp. 312-326, 2006

[24] Minkenberg, C., M. Gusat, "Congestion management for 10G Ethernet," *Proc. Second Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip (INA-OCMC 2008)*, Göteborg, Sweden, Jan. 2008

[25] Agarwala et al. "SysProf: Online distributed behavior diagnosis through fine-grain system monitoring," *Proc. Distributed Computing Systems, 2006. ICDCS 2006,* pp. 8-8, 2006

[26] Chanda et al., "Whodunit: transactional profiling for multi-tier applications," *SIGOPS Oper. Syst. Rev.* vol. 41, pp. 17-30, 2007

[27] Jun Li, "Monitoring and characterization of component-based systems with global causality capture," *Proc. 23rd Int'l Conf. on Distributed Computing Systems*, pp. 422-431, 2003

[28] Gschwind et al., "WebMon: A performance profiler for web transactions," *Proc. WECWIS 2002*, pp. 171-176, 2002

[29] Application Response Measurement, http://www.opengroup.org/management/arm/

[30] TPC-W benchmark, http://www.tpc.org/tpcw/

[31] Schmid et al., "Measuring end-to-end performance of CORBA applications using a generic instrumentation approach," *Proc. ISCC 2002*, pages?

[32] Lin et al., "A new TCP and UDP network benchmark suite". *Proc. 2007 Spring Simulation Multiconference* – Vol. 1, pp. 211-217, 2007 [former 39]

[33] Meijer et al., "User programmable virtualized networks," *Proc. Second IEEE Int'l Conf. on E-Science and Grid Computing*, 2006

[34] Suzumura et al., "Performance Comparison of Web Service Engines in PHP, Java and C", *Proc. 2008 IEEE Int'l Conf. on Web Services*, pp. 385-392, 2008

[35] Sachs et al., "Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark," *Perform. Eval.* (2009, in press).

[36] Lam et al., "A performance study of clustering web application servers with distributed JVM," *Proc. Parallel and Distributed Systems, 2008. ICPADS '08*., pp. 328-335, 2008

[37] Traeger et al.,. "A nine year study of file system and storage benchmarking," *IEEE Trans. Storage*, pp. 1-56, 2008

[38] Phansalkar et al., "Four Generations of SPEC CPU Benchmarks: What has changed and what has not?" Technical Report TR-041026-1, The University of Texas at Austin, 2004.

[39] Lu, Y., et al.,"Congestion control in networks with no congestion drops," in *Proc. 44th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Sept. 2006.

[40] Minkenberg, C., et al., "Adaptive routing for convergence enhanced Ethernet," *Proc. 2009 Int'l Workshop on High-Performance Switching and Routing (HPSR 2009)*, June 2009, Paris, France (to appear).