RZ 3743 (# 99753) 12/21/2009
Computer Science 7 pages

Research Report

Transaction Tracking in Large-Scale Datacenters

G.J. Paljak*‡

* IBM Research – Zurich, Rüschlikon, Switzerland

‡ Budapest University of Technology and Economics, Budapest, Hungary

Transaction Tracking in Large-Scale Datacenters[1]

G. J. Paljak
IBM Research, Zurich Research Laboratory
Budapest University of Technology and Economics, Hungary
gpa@zurich.ibm.com

Abstract— The current evolution of over-provisioned, under-utilized datacenters (DCs) to resilient, dynamically reconfigurable, virtualized cloud computing DCs that are managed to meet optimums in terms of performance and energy consumption requires new analysis and benchmarking methodologies, which poses significant challenges in system monitoring and modeling. We present a technical view of business transaction tracking in DCs, and its uses in analytical and simulation modeling.

I. MOTIVATION

Cloud computing is a new way of providing IT services; in a cloud-computing DC, applications are provided as standardized offerings to end users over a dynamically reconfigurable system of network and computation nodes. This requires significant transition from traditional ways how computational resources were used, benchmarked, and optimized.

The traditional DC has taken advantage of the low cost of computing power and often evaded performance issues by over-provisioning the system with a high quantity of under-utilized servers, storage, and network capacity, 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.

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. Scale-out and virtualization leads to separation of functionality. It is uncommon for a virtual machine to provide multiple services (as opposed to a mainframe system), instead a service is often provided by a set of virtual machines (cluster of many identical machines). Functionality is federated on the level of virtual machines running on the hypervisor (managed environment, ensemble management). Virtualization also provides the inherent advantage of task migration, replication as a baseline service. Therefore, it is vastly widespread that the processing of a client request spans through multiple machines and multiple applications. Our aim is to analyze the performance of such systems by tracking the transactions in the heterogeneous environment.

While performance optimization on a component level for individual servers is well developed, it is definitely less mature on an integrated systemic level. The workloads are heterogeneous, asynchronous, and not well understood at the DCN level.

Traditional component-level benchmarks include the SPEC benchmarks often used for CPU and cache profiling [24], or network component benchmarks, such as [22]. Often analysis focuses on examining the performance of NICs and network protocols [32]. However, efforts such as TPC-W benchmarks for Online Transaction Processing (OLTP) [30] intend to measure 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^{TM} 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. In this paper, we study a technical aspect of DC performance analysis, tracking transactions in a distributed system. We address the problem of DC-level modeling in both analytical and simulation means.

The paper is organized as follows: in section II. we describe the concept of DC modeling, in II.A we give an abstraction of DC topology, in II.B the importance of transactions and traces is described, in II.C we sum up some analytical work on workload benchmarking, in II.D. we present a simulation methodology effectively used in large-scale systems, and in II.E summarize the challenge of transaction-tracking. In section III.1 we give an overview of efforts on transaction tracking in distributed systems, intended as a tutorial, and III.2. our approach is presented. Section IV briefly outlines our future work.

[1] An abridged section of this report will be published in [2]

II. A FRAMEWORK FOR DC ANALYSIS

A. A description of the DC topology

To describe the structural configuration of a DC, 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. Mappings between the three graphs exist: the 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. Business transactions as paths on the topology

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 occupancies: 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 3).

In order to have a record of such a transaction, its events, the data being passed is generally described in the following format (referred to as trace format): {(1) TimeStamp | (2) SRC | (3) DST | (4) Prio | (5) BSize }. In order to maintain the causality relation, it is further extended with the *(6) GlobalID* provided by transaction tracking module. In order to preserve the causality relation among events, a central database of the GlobalIDs is maintained: this is typically a tree that where a parent-child relation indicates causality.

C. Analyisis of DC workload and metrics

The application transactions mapped to the network layer translate into a trace of flits, packets/frames, and flows. 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).

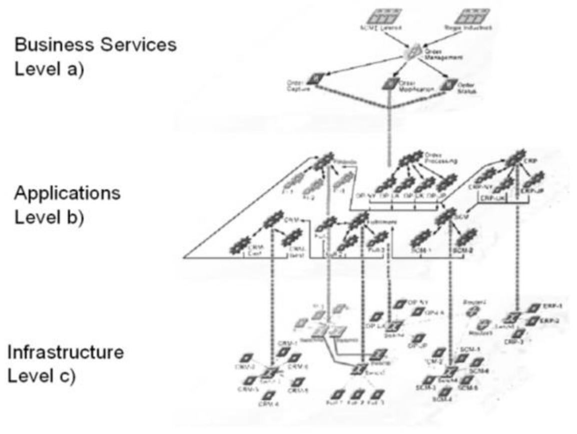

Figure 1. Abstraction levels of a DC structure

The best-established (verified by both analytical and simulation means) 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 (Datacenter Bridging) 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 $FCT = \Sigma\,(t_{\text{inject}} + t_{\text{queue}} + t_{\text{flight}} + t_{\text{RTX}}) \neq L_{\text{e2e}}(X)$, 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, 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 aggre-

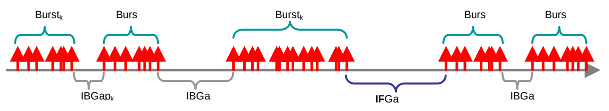

Figure 2. Transaction consisting of 5 bursts, each having 7-15 Ethernet frames, with an arbitrary burst size and inter-burst gap.

gate statistics. The translation between application and DCN (L7:L2) metrics is the role of “integrated” metrics conversion—a problem yet to be solved.

Also, the burstiness of workloads on the application level is studied in [5], authors emphasize the importance of the bursty nature of workloads and claim that it has primary importance on overall performance, as it may lead to a different general modes of operation (bottleneck switching).

Based on these, we propose the trace-based in-depth transactional analysis of DCs, to be able to detect such situations and to trigger proactive re-configuration.

Traces are also widely used for empirical validation of analytical models, like in processor design [4], wireless networks [3], or High-Performance Computing (HPC) systems with tens of thousands of nodes, described in II.D.

D. VENUS: an OMNeT++-based large-scale simulation platform

Traces are also widely used in a trace-driven simulation. In those cases, 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 the Message Passing Interface (MPI), tracing MPI calls (by instrumenting MPI libraries) is a suitable method to characterize the communication patterns of HPC workloads.

An example of this approach is the MARS simulator presented in [18], or of its successor, the VENUS-Dimemas HPC tracing and simulation environment, depicted in Figure 3. Both of them are based on the OMNeT++ discrete event simulator [38].

The co-simulation environment composes of VENUS responsible for detailed simulation of the net-

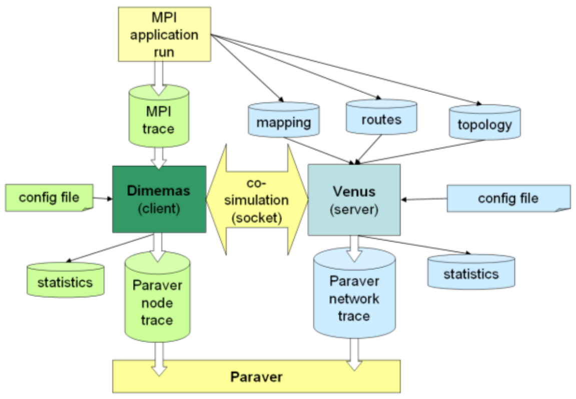

Figure 3. HPC co-simulation platform: VENUS/Dimemas

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

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 causally ordered sets of communication and computation primitives.

E. Why is transaction tracking difficult

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

(i) The lack of a de-facto standard DC communication protocol similar to the 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 if one has an instrumented version of some of these protocols, generalization is still an open issue.

(ii) Observing and rebuilding causal paths (trajectories) are complex 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.

In Section III we present methodologies to overcome the challenges.

III. Transaction monitoring

A. Approaches for transaction tracking

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 building and validating analytical models, or driving the simulation.

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 instrumented. Throughout the implementations of this concept, either application-specific assumptions, or globally unique identifiers (GUIDs) are used.

This is the case for NetLogger [9] or the Application Response Measurement standard (ARM, [29], Figure 4) where the application source code is extended with explicit tracing information. NetLogger provides near real-time analysis and anomality detection. It requires its instrumentation code to be inserted to the application source code (similarly to ARM), but NetLogger focuses on matching start and end events of tasks, and does not handle correlation across a distributed transaction (unlike ARM). ARM is an interface specification designed for monitoring transactions with the ability to handle correlation IDs; the interface specification and free or commercial implementations are available for several programming languages, including C and Java.

WebMon [28] uses HTTP cookies modified for storing GUIDs that are created by custom JavaScript, and are passed to instrumented web- and application servers. The fact that correlation is done on HTTP level leads to a coarse-grained component definition, meaning less insight to the system (e.g. no performance information on database queries or other sub-transactions provided), but also less overhead.

User Programmable Virtualized Networks, described in [33], provide the developer with the freedom to handle network interactions and take some of the generally OS responsibilities. This requires one to extend the application and to modify the typical OS kernels' networking stack; and offers the opportunity of insert tags with IDs into the packets and handling those on the level of the custom application/OS component.

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, protocol implementations, middleware applications or runtime environments to support tracing. This approach is transparent towards the application.

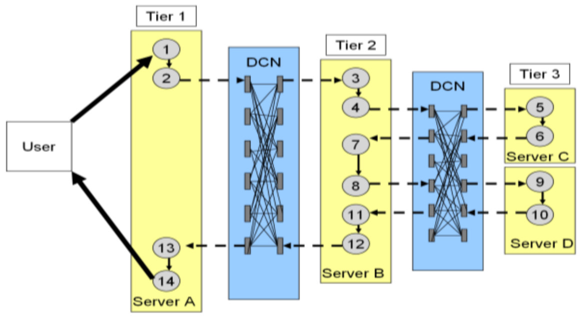

Figure 4. The ARM projection of a transaction on the DC infrastructure

Next, we look at some *gray-box solutions*, Magpie [10] is built on Microsoft® Windows® platform and uses a built-in event-logging framework (Event Tracing for Windows, ETW) extended also by custom, middleware-specific event generator code. They use an event schema for reconstructing causal paths; this externally added platform- or application-specific knowledge describes which attributes connect events to each other to form a path. (Earlier versions of Magpie used GUIDs.) PinPoint [11] uses platform-specific tracers that extend protocol implementations and middleware, such as the web server, J2EE containers, the JDBC and the Remote Method Invocation (RMI) protocols, to tag transactions with GUIDs and to preserve the tagging. A centralized aggregator reconstructs the correct paths and stores in a repository for further analysis, visualization. However, they do not track lower level events (e.g. network packets), which, for example, Magpie does.

A similar instrumentation in [12] for the Java Virtual Machine uses agent-injection to the runtime-environment and inserts trace statements ahead of the flow control, and support propagation through TCP with additional platform-specific instrumentation. The path reconstruction is similar to that of Magpie, harnesses application specific knowledge, not GUIDs, to drive a series of graph transformations.

Common Object Requesting Broker Architecture (CORBA), being a well-accepted standard for distributed systems, has transaction-tracking methodologies. Its built-in concept of interceptors (interrupting the request flow between a client and a server and executing custom code) provides useful facilities for this purpose. Authors of [31] take advantage of these interceptors and use them to insert ARM-compliant transaction tracking code the control flow on the middleware layer. In [21] as well, interceptors are used for trace collection, in this case caller-callee pairs, not multi-stage transactions. A statistical analysis system for processing the traces was also developed. For CORBA/COM, a method is introduced in [27] that uses automatically generated (by the IDL compiler) skeletons and stubs to identify first caller-callee relationships and then to propagate GUIDs to observe transactions spanning across multiple components.

In [13], ISA level instrumentation is used, i.e., a four-byte tag is maintained for every byte of the memory and the propagation of tags is assured by inserting into and extracting from Ethernet frames. It is also possible to add specific tagging from the application source code; this can be done by the C library developed by the authors. In languages like Java where the virtual addresses are hidden, such application-level tagging becomes difficult to achieve. This kind of instrumentation requires profound modification of OS components, device drivers. Whodunit [26] detects transactions communicating through shared memory, events or RPCs by means using OS-level instrumentation (creating wrappers for several functions, e.g. *send* and *receive* wrappers to follow IPC), and running critical sections in QEMU, a CPU emulator. The overhead of OS instrumentation is approx. 3%, but the execution on the emulator is 100-400 times slower, according to case studies. SysProf [25] is a profiler, keeping track of resource utilization using kernel-level instrumentation, however it is capable of supplying transaction profiles, it requires additionally supplied domain-, or transaction-specific knowledge to identify the causality of transactions when they are interleaving or concurrent.

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 can only be based utilizing the events and the timestamps of the transaction, that can be easily logged, by probabilistic and statistic means, therefore fully correct transaction-instance level causal path reconstructions are not feasible. Authors of [16] show a model for corresponding events to re-construct a transaction by finding matchings in bipartite graphs; optimal solution for two-state systems with independent, identically distributed transition times corresponds to a minimum-weight perfect matching in the graph. In [15] the authors consider a similar approach and 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-correlating timestamps of the messages between infrastructure components. In [17] and [20] a network-only approach focusing on TCP is introduced, they do not offer transaction reconstruction, but coarse-grained application-level interaction discovery. In [17] authors profile the network usage of applications and correlate it to resource consumption. [20] is also an enhancement to network packet sniffing by mapping interconnected TCP sessions using user-supplied domain knowledge. In [23] the authors do not intend to identify the causality relation, but they study the mass characteristics of transactions and describe flow dynamics in some monitoring points (e.g. Apache HTTP logs, MySQL query logs) with autoregressive models. The fault detection scheme they propose determines invariants (of the mass characteristics) and identify faults when these invariants are broken.

We must also call the attention to three main challenges for black-box approaches: (i.) they heavily rely on well-synchronized clocks, but claim that the 1-5 ms error of the Network Time Protocol (NTP), the implementation of which is generally available as a built-in OS component, is satisfactory; (ii.) asynchronous calls are harder to trace without additional knowledge, than request-response interactions; (iii.) in case of kept-alive connections or sessions (typically from a connection pool) one cannot make use of connection opening and closing events as start and finish events of (sub)transactions.

B. Orchestration prototype: a grey box scenario

In our research, we stated the following requirements against transaction tracking: (i.) avoid modifying application source code (as it is rarely accessible in a production environment); (ii.) emphasize correct causality re-construction; (iii.) adhere to standards, as much as possible.

Therefore, we decided to 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 [19]. This software is designed for monitoring distributed systems, transaction tracking is based on the ARM standard. Its main advantage is the middleware-level support from multiple vendors who natively support automatic insertion of ARM calls to the control flow of the application and harnessing native middleware instrumentation. However, general use cases include non-supported middleware, in that case monitoring is still possible by extending the specific application with ARM-compliant instrumentation code, according to [29], or adding ARM support to the middleware, like in [7]. ITCAM 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 4).

IV. CONCLUSIONS AND NEXT STEPS

We have shown uses and the challenges of DC transaction tracking, and some of the methods to overcome these challenges. 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 [8], and analytical modeling of dynamic and distributed systems.

Next Steps: With the progress toward cloud computing 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. 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 tools* capable of (DC and cloud) system-level benchmarking. The latter lack is sustained by the continuing prevalence of segregated, component-level benchmarking. Nonetheless, despite these challenges, DC and cloud management and optimization are a promising area of research for the systems community.

V. ACKNOWLEDGEMENTS

We thank M. Gusat, C. DeCusatis, C. Minkenberg, L. McKenna, K. Bhardwaj R. Luijten, W. Denzel, G. Rodriguez, A. Pataricza, I. Kocsis, Z. Egel, and D. Toth 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] M. Gusat, C. Minkenberg , C. DeCusatis, L. McKenna K. Bhardaj, G.J. Paljak, A. Pataricza, I. Kocsis , “Benchmarking the Ethernet-Federated Datacenter”, First Workshop on Data Center - Converged and Virtual Ethernet Switching (DC CAVES), Sept. 2009, to appear

[3] Eckhardt, D. A. and Steenkiste, P. A trace-based evaluation of adaptive error correction for a wireless local area network. Mob. Netw. Appl. 4, 4 (Dec. 1999), 273-287, 1999

[4] Ho, R. C., Yang, C. H., Horowitz, M. A., and Dill, D. L. Architecture validation for processors. SIGARCH Comput. Archit. News 23, 2 (May. 1995), 404-413, 1995

[5] N. Mi, G. Casale, L. Cherkasova, and E. Smirni, "Burstiness in Multi-Tier Applications: Symptoms, Causes, and New Models", Middleware 2008 , Leuven, Belgium, December 2008, Lecture Notes in Computer Science, Volume 5346, pp. 265-286, Springer Berlin / Heidelberg, 2008

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

[7] ARM Instrumentation Framework for JBoss and Tomcat, http://wwwvs.informatik.fh-wiesbaden.de/oss/jboss-arm/index.html

[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] Tivoli Composite Application Manager for Transactions http://www-01.ibm.com/software/tivoli/products/composite-application-mgr-transactions/

[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] 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.

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

[24] 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.

[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

[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, vol. …pp. 1-56, 2008

[38] A. Varga, “The OMNeT++ Discrete Event Simulation System”, Proceedings of the European Simulation Multiconference (ESM 2001), June 6-9, 2001, Prague, Czech Republic.

[39] QEMU, open source processor emulator, http://www.qemu.org/