Grok the data center

Zaheer Chothia, Qin Yin, and Timothy Roscoe
Systems Group, Department of Computer Science, ETH Zurich

Abstract

In this paper, we claim that a vital part of managing a large enterprise data center is creating a model which encompasses all layers of the hardware and software stack, from network links up to application placement and configurations, and moreover that this model must be maintained in real time based on monitoring and trace data. Such a claim is not necessarily contentious, but we further argue that existing approaches to managing such facilities suffer from a lack of abstraction of the problem that limits their applicability: enterprise data centers are sufficiently diverse that it is hard to identify a reusable component that can express the complex range of queries that must be served in real time. The problem is less that of working out what data to gather, and how to process it, than it is one of building a flexible, reusable infrastructure above which to perform such processing. We therefore characterize the problem, and the systems challenges entailed by such a "RDBMS for system management". Finally, we report on our early exploration of the problem space, showing that even a limited version of such an engine can deliver useful value to a real, large-scale, production data center.

1 Introduction

Enterprise data centers are complex, multi-layered, evolving software and hardware systems whose behavior is hard to understand, but whose stability is critical to the success of an enterprise. In this paper, we argue for building an online, cross-layer model of an operational enterprise data center as a separate, critical component of a disciplined systems approach to operations and management for the data center. Such a model integrates information from physical topology up application-level message traces. We further argue that maintaining such as model constitutes a workload not well served by current information processing software stacks, and is therefore an important research challenge in itself.

By *enterprise data centers* we mean large-scale computing facilities operated internally by a large enterprise, in contrast to *cloud computing* centers where compute resources are rented out to paying customers. Most of the world's data centers fall into the former category, although some of the approach we advocate is also relevant to the cloud computing case. We elaborate on the differences between these use-cases below.

By *online, cross layer models* we mean representations of the state of the data center which are updated in real time and integrate system configuration, real-time monitoring, and trace data from all levels of the software and hardware stack, from physical network links and servers up to application components running on specific physical and virtual machines.

Datacenters in large enterprises are usually the result of a complex ongoing evolutionary process driven by constantly changing goals. Consequently, they contain a heterogeneous mix of machines, from commodity servers to mainframes, and a highly structured network reflecting organizational boundaries within the enterprise as well as resilience, efficiency, and manageability concerns. Many applications are tied to subsets of machines for a variety of legal, contractual, and technical reasons. They also have elaborate inter-dependencies, often encoded in a "Service-Oriented Architecture" framework that ties them together. Moreover, this is a dynamic environment with configuration and software changes occurring on a frequent basis.

Managing and evolving such a datacenter is therefore a complex, labor-intensive process requiring considerable

APSys '14, June 25-26, 2014, Beijing, China

skill and experience, particularly as the availability and performance of these applications is critical to the success of the enterprise. This also presents a challenge to accurately model and analyze such a datacenter. From a systems research perspective this also makes an interesting use case: the tooling should support a high volume of monitoring and tracing data updates, queries involves a mix of relational, graph and stream processing, and there is a desire for results to be generated within a timescale of seconds to minutes.

We focus on the problem of building and maintaining a faithful model that can answer queries about the behavior of applications and infrastructure in the datacenter, and avoid the separate question of what management action to take at a given moment. However, we believe that a flexible, expressive, cross-layer model of the datacenter forms a sound basis for active management decisions.

We also restrict (for now) the model information to that from the networking and computer hardware and software in the datacenter, eliding information about power, temperature, etc. We acknowledge, however, that the latter are ultimately important components of such a model.

Where we depart from prior work is our emphasis on the construction and maintenance of a model of the state of the datacenter. We are inspired by, and build on, considerable existing work on data centers which has explored the kinds of questions important to both management, and integrating information from multiple sources.

Our goal is to develop a common *infrastructural engine* for online modelling of a datacenter, in much the same way that a RDBMS has traditionally been a common infrastructure for enterprise data. One way to view our work is to design a single, rack-scale machine which can process the complex range of queries that must be served in real time.

Our contributions in this paper are:

- Identifying data center modelling as a useful factoring of the problem of managing a data center.
- Characterizing the workload a model must handle.
- Presenting early experience showing the value of such a model in the context of a real-world datacenter belonging to a large travel industry service provider.

2 Background

Operating a production data center at scale is a tall order. A large enterprise data center runs distributed systems built of many interacting components. These systems have complex dependencies and undergo frequent change. They impose elaborate demands on networking and computational infrastructure. Operators need to ensure that demanding performance requirements are met, and must resolve incidents in a timely manner with minimal customer impact.

The datacenter we use to evaluate our work had the following characteristics in 2012 it has subsequently expanded considerably:

- 19000+ transactions per second (peak)
- <0.5 sec average system response time
- 9000+ physical IT infrastructure devices
- 3400+ IT changes per month
- 400+ application software loads per month

This is the main data center for a large provider of services to the travel industry, but is representative of a large class of enterprise data centers which host multiple web services and applications. Advertising platforms are also highly-concurrent, serving 10 billion events per day, and latency-sensitive, with a requirement to respond within 100 milliseconds without fail [9]. In these facilities, "rare events happen frequently" and "bad things transpire faster than humans can respond" [9].

We argue that the field can only move forward if we adopt an approach which abstracts the problem: we aim to build a system which is flexible enough to adapt to the complex and diverse mixes of monitoring and tracing infrastructure found across enterprise data centers, and yet is efficient enough to provide, in real time, the detailed information and analysis needed by both human and automated management.

3 Related work

Unsurprisingly, there are numerous commercial and research attempts to address aspects of this problem.

Many target only a small subset of the hardware/software stack: either network- or service-layer. One typical example is Google System Health [3], which monitors servers and their configuration, activity, environmental and error data, and conducts offline analysis. Similar monitoring and analysis tools exist for networking infrastructure, such as CiscoWorks or HP OpenView. Internet service providers also deploy online monitoring of key signals such as latency and throughput statistics for user requests. Operator alerts are then issued based on monitored values and predefined thresholds.

Another class of tools tackle specific aspects of the problem: performance debugging. Sherlock [1] applies a *black-box* approach which runs traceroutes to learn wide-area network topology, deploys agents to capture traffic, and statistically infer causal relationships to localize failed network components. NetMedic [7] harnesses information exposed by applications and OSes to infer correlation between finer-grained network components such as processes. Such systems are the ancestors of recent startups such as New Relic and CloudPhysics, which mine data from application-level logs and event traces in order to diagnose performance anomalies and issue alerts.

Recognizing that many of the hard-to-diagnose faults encountered in enterprise data centers involve interactions between disparate layers of the software and hardware stack, some systems adopt a more cross-layer approach.

Pinpoint [4] and Magpie [2] are early examples of such systems which target the debugging and profiling of individual applications. Pinpoint traces client requests and failures and diagnoses faults with offline analysis. Magpie records a wide variety of information about web requests (including resource usage) which forms the input to offline performance or root-cause analysis. X-Trace [5] is a cross-layer, cross-application tracing framework designed to reconstruct the user's task tree.

We are not the first to note that many of the performance and fault diagnosis problems seen in operational datacenters can only be solved with a comprehensive cross-layer approach. At the same time, however, there is a dearth of such systems available to enterprises: commercial (or, indeed, free open source) offerings tend to operate on a small subset of the hardware and software stack, and those few enterprises which have tried to implement a cross-layer solution have done so at considerable cost in engineering.

This mismatch between readily available technology and the problem at hand is partly due to the organization of many enterprises, which separate (for example) network management, server provisioning, and application maintenance into different units.

However, the biggest *technical* barrier is the scale of engineering necessary, coupled with the lack of portability: enterprise data centers and their software vary widely, and the solutions we survey in section 3 targeted a single deployment. This situation mirrors the state of enterprise data management in the late 1960s, before the advent of general-purpose relational database managment systems.

The work listed above differs from ours in two ways. Firstly, our system aims at *cross-layer* datacenter modeling and analysis which considers all layers of the hardware and software stack, using dynamic tracing as well as static configuration.

Secondly, prior systems systems focus on tracing and workload modeling, and run performance analysis as offline jobs. We provide a general computing framework for both datacenter modeling and performance analysis online for various involved applications. This can potentially be integrated to the alert system to provide real-time and more sophisticated diagnoses.

4 Systems challenges

The long-term goal of our research, then, is to create a data model and data management system which is capable of expressing, maintaining, and updating a detailed, cross-layer, online model of a data center. The system must be capable of absorbing large-volumes of real-time monitoring and trace data (including frequent configuration changes to the data center) while performing useful analysis on the model in real time.

In this section we discuss the systems challenges and our initial work investigating the workload such a system must handle; following this we elaborate on the long-term goal mentioned above.

4.1 Building a Model

From the perspective of building a model, there are a number of issues to contend with. A few are discussed below and these primarily relate to the inconsistencies, incompleteness and sheer complexity of the data at hand.

Gather and reconcile: In a large enterprise, knowledge is generated and maintained by different departments across the organization, and therefore potentially distributed in various geographical locations. Different teams work on different levels of abstraction and use different representations of the underlying infrastructure. Such information fragmentation and incoherency poses a great challenge to integrate disparate data sources and build a comprehensive model of the datacenter using information from all the layers of the hardware and software stack. In particular, this entails gathering all the requisite data, understanding its semantics and reconciling potential inconsistencies.

Store and query: The next question to answer is how to represent, store and query state of the datacenter in real-time. Such a knowledge base forms the foundation which further analyses build upon. The datacenter information sources exhibit high heterogeneity: some data (such as network topology and system configuration) is of small volume and evolves more slowly, while the other (like real-time dynamic workload traces) has larger data volume and higher update rate. Despite storing and querying heterogeneous data, there are also further questions relating to the large volume of on-the-fly data: which tools are suitable to process the data and how much archive data should be retained?

Complexity and dynamism: Beyond this, further considerations arise such as those of model fidelity and complexity. The available data spans several layers of the stack and covers a multitude of abstractions. What information should be included in the model and how much detail needs to be preserved? It may be tempting to feed in as many data sources as feasible, but there is also a trade-off with generality. A good model should also be applicable in different contexts. This requires capturing the key primitives and also factoring out aspects specific to a given setting. A datacenter is also a very dynamic environment: nodes fail, configurations change and new infrastructure is deployed. This continual change also affects the model and makes it especially hard to pinpoint the true source of an error.

Data volume: It is difficult to maintain a live model in the presence of a high update rate, whilst also conducting analytic queries, even more so, if the state is distributed across multiple machines. Much of this difficulty lies in the workload model which comes from middleware logging. This data is sizeable – several terabytes per hour – and has a tree structure. Graph databases make a good fit, but many of these cater primarily towards large-scale social graphs which have very different properties and looser consistency requirements. In general, two characteristics we foresee as necessary are distributed and in-memory execution. The former allows to exploit the available parallelism within a rack of machines; the latter allows to store large quantities of data and process with minimal latency. The Spark stack [6] is based on this premise and therefore makes a promising foundation.

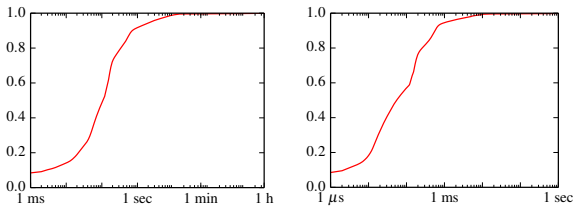
The ultimate goal is a system for on-line modeling (data representation and storage) and analysis (data processing) of a datacenter based on configuration and trace information from multiple layers of the hardware and software stack. Such a facility should support answering complex analytic queries about the performance of the datacenter under both real and hypothetical conditions and workloads. Next we elaborate on available data sources and conduct a case study of a production workload.

4.2 Workload properties

A guiding principle of the desired system is the integration of information from as many layers as possible in the datacenter. Fortunately, the datacenter we study already has extensive instrumentation and logging and the available data comes from two main sources: *monitoring systems* and *tracing systems*.

The former has agents embedded throughout the datacenter and collects metrics from the observed systems, including hardware and software components. This primarily includes performance indicators, such as network link traffic, CPU utilization, queue lengths, etc. The latter collects targeted events from the systems being observed, for example the messages being sent and received and system calls executed, and can be employed to trace requests as they flow through a distributed system. Further complementing this are various *configuration databases*, which store a vast swath of information: application deployment and placement, routing and switching tables, along with a history of configuration changes (e.g. commands executed). The key challenges are integrating the disparate information sources (which, in some cases, contain inconsistencies), deciding what information to include in the model, and how it is represented.

In order to clarify the requirements of the envisioned system we now conduct a case study on a 57-minute detailed OLTP workload trace. Our trace involves 7118 machines (a significant fraction of the entire datacenter),



(a) Transaction Tree Duration (b) Maximum Inter-arrival Time

Figure 1: CDFs (Cumulative Distribution Functions) showing total duration of transaction trees and maximum interval between messages of a single session. Note: the x-axis is in logarithmic scale.

and consists of about 197 million individual transactions, corresponding to roughly 47 million externally visible operations (transaction trees) belonging to 30 million client sessions. This datacenter processes in total more than 4 billion travel transactions a day.

To give a sense of the problem size and requirements on the corresponding systems we now present some figures about the workload trace. The vast majority – roughly 95% – of all root transactions are short-lived and have a total lifetime of less than 2 seconds (Figure 1a). Similarly, in only 0.24% of cases do root transactions remain dormant for more than a minute (Figure 1b); this means a streaming system does not need to retain a large quantity of state in working memory while correlating messages.

5 Goals

Having discussed challenges in building an online, cross-layer model and data management system, we now expand further on our goals and potential applications.

A general model

We aim for a *holistic view* of a data center which integrates hardware, network, middleware, and application information, providing a coherent model which captures interactions and effects across layers. This should be based on continuous monitoring of the target system.

In the early exploratory work we present in Section 6, for example, we collect network topology, switch configuration, application and virtual machine placement across physical machines, and a detailed trace of application-level messages and transactions.

It should be pointed out that none of this data is ultimately static. For example, in the data center we target there can be several hundred changes to network and server configuration each day, and each logged change can affect many components. It is therefore insufficient to assume (as we temporarily do in our current system) that the network, for example, remains the same.

We also have to deal with high incoming data rates for some kinds of data. For example, we trace all messages

sent and received by application routers which form part of the service-oriented architecture (SOA) of the data center. These traces are used to reconstruct *client sessions* as sequences of *transaction trees*, analogous to "trees of spans" in Google's Dapper system [8]. This constitutes several terabytes of trace data a day alone, and must be processed in real time.

Wide applicability

We aim to use the data collected to answer complex continuous queries involving considerable integration of information, such as tracking call patterns or dependencies between services within a distributed system, mapping application-layer traffic down to physical network links, and correlating events across the data center.

The queries we therefore pose use a combination of processing paradigms: node and application configuration involve joining and grouping relational tables, whereas network path calculation involves graph traversal and shortest path queries. Streaming and CEP (complex event processing) systems are needed to maintain the online model. We now give some concrete examples, inspired either by existing systems or the requirements of the data center we discuss in Section 6. There is a wealth of prior work in these fields and [10] is an extensive survey of data center troubleshooting, instrumentation and tooling; we now discuss a selection of potential applications.

Detection: identifying potential outages or performance degradation by monitoring the system for SLA violations or abnormal states. Some problems are subtle and difficult to detect, e.g. quantifying the proportion of application traffic that traverses specific network layers, as we evaluate in the next section. In another, a web application may return `200 OK` but serve meaningless results. Operators have limited knowledge about application behavior and a cross-layer view is needed to understand this. A key challenge here is that simple metrics and thresholds also do not suffice, and often cause 'alarm storms' where an operator is inundated by correlated failures.

Diagnosis: analyzing anomalies in order to determine their root cause, or suggest solutions. One theme is to mine archive data and build a model of unhealthy symptoms or system states. Another theme is to understand the relationships between components, for example service dependencies or transaction tracking. This is of use when troubleshooting communication issues and localizing problems as they flow through a distributed system.

Simulation: the effort and risk associated with change can be prohibitive in a production data center. It is thus desirable to have a testbed to induce failures or try changes before they are rolled out. Examples include investigating the effects of new routing protocols or testing topology-aware load balancing. The problem is that a testbed which functions at scale is beyond the means of most enterprise data centers. However, a high-fidelity model driven by real workloads can be used to analyze such "what-if" scenarios, an idea we explore in the next section.

6 Exploratory prototype

To investigate the systems challenges of building a modelling system for a datacenter, we started with a simple prototype using off-the-shelf components. Figure 2 shows the architecture; it includes components for data integration, cross-layer modeling, data processing, and cross-layer analysis.

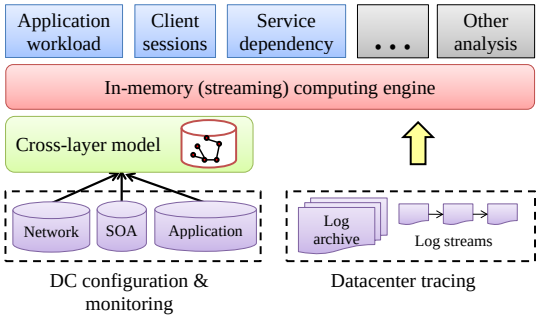


Figure 2: Prototype system architecture

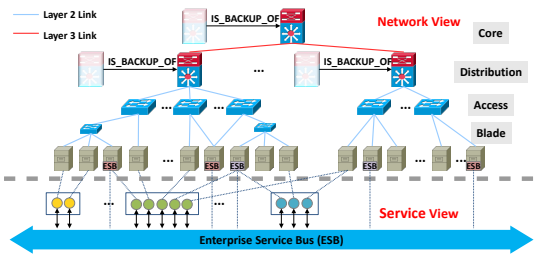


Figure 3: Data center topology model

This system does *not* address the requirements in section 5 – it assumes a static network and application placement, and runs offline on an hour-long trace of application messages – but we have found it useful as a first step in exploring the design space for data center modelling.

The specific data center we apply it to is the travel service provider mentioned in section 2. The architecture (at the time of the experiments reported here) was the fairly traditional enterprise design shown in Figure 3. We are fortunate in that the data center is heavily instrumented, and we can obtain logs of all application-level messages sent and received by the applicaton level "enterprise service bus" (ESB) routers, along with their timestamps, which make up the OLTP workload the data center.

A rich data representation is constructed from these data sources. At the network level, we obtained a snapshot of the physical connectivity of the datacenter, along with IP, MAC addresses and VLAN membership for each

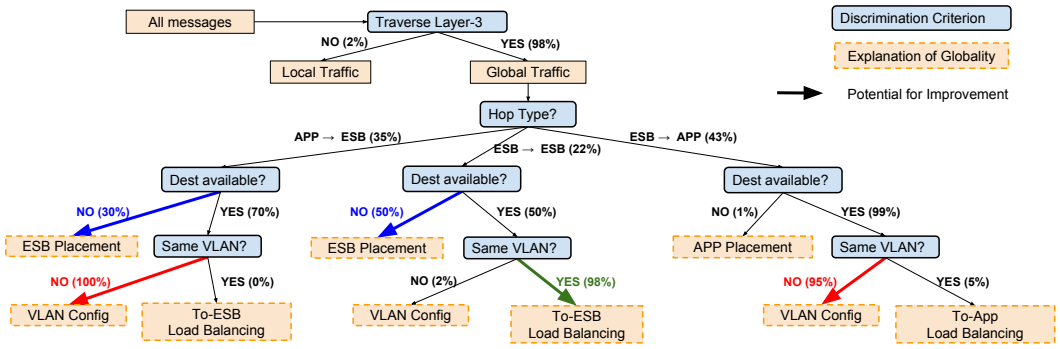


Figure 4: Detailed reasons for hops traversing the distribution layer

machine. From this, we derive a multi-level graph representation of the datacenter interconnect at layers 0-3 as shown in the upper part of Figure 3. This network view provides us with detailed information about switches, all physical and virtual machines in the data center.

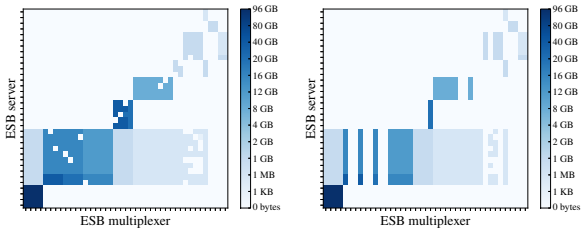
We also incorporate the configuration database giving the current state of the ESB, including service publication, grouping services into applications, application servers, ESB machines, ESB processes running on the ESB machines and other lower-level entities. This service view is depicted in the lower part of Figure 3. We use the JGraphT library to create in-memory annotated graph objects to represent network topology and to traverse graph nodes and links while we keep most middleware and application configuration information in a RDBMS.

Our OLTP trace information is a 57-minute trace from which we reconstruct *client sessions* composed of *transaction trees*, as in Dapper. This trace represents about 4TB of data traversing the network. We process this trace using Spark [6], and replicate the (relatively small) graph database to all Spark nodes.

Even with this relatively simple system, useful results can be obtained with simple queries. For example, the company suspected that a large proportion of traffic was traversing the core and distribution layers of the network unnecessarily, but could not quantify this.

Figure 4 shows the results of an analysis mapping all application-level messages through the multiple layers of the protocol stack down the layer 0 to calculate the physical path taken by each message. From this the system calculates not only the proportion of traffic traversing the distribution network, but also *why*: factors include placement of the ESB application-layer routers, the default round-robin load-balancing policy at the ESB layer, and the current VLAN configuration of the network and the deployment of different components in different VLANs. This information was of great value to the company.

Moreover, given a detailed data center model, we can go further and investigate "what-if" scenarios to quantify



(a) Baseline (b) Topology-aware

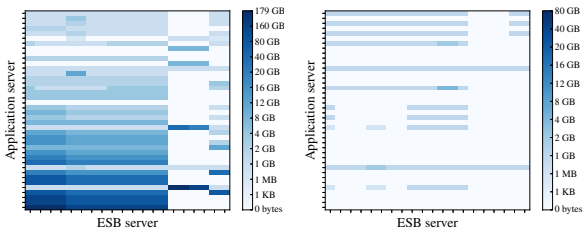
Figure 5: ESB–ESB traffic

the effects of configuration changes to the data center. The first example we present concerns changing the load-balancing policy used to select an ESB router from round-robin to one which is aware of the underlying network topology, and always picks a nearby instance if it can.

The heatmaps on the left of Figure 5 is generated from the model and shows traffic between ESB routers and multiplexers in the system, clearly showing the grouping of such components into evenly load-balanced "farms". The right map shows what the result of changing the policy would have been: several strips are now removed because the communicating ESB pairs are more likely to be located under the same access switch. This strategy helps to localize almost 1TB of traffic from the total 3.75TB distribution traffic. Our system achieved this result by changing the routing routine and re-running the analysis.

Our second "what-if?" example concerns traffic between applications themselves and the ESB routers. Figure 6a shows a portion of the heatmap of ESB-application communication traffic (the full traffic matrix is large, but this subset represents over 90% of the distribution traffic). Almost all this traffic traverses the expensive distribution switches (10.25TB out of 10.31TB in our trace).

The main problem here is that the ESB servers reside on different VLANs, which means traffic has to be pass through the distribution layer to be routed at Layer-3. We can investigate the effect of placing both ESB routers



(a) Baseline (b) With single VLAN

Figure 6: ESB–APP traffic

and application servers into a single VLAN, allowing messages to be routed to the local application replica or ESB service if it exists. Figure 6b shows the heatmap of this traffic after removing VLAN difference and applying a topology-aware communication strategy. This naive elimination of VLANs dramatically reduced the global traffic from 10.25TB to 430GB.

In practice, of course, security and organizational concerns might rule out such a simplistic approach, but the potential saving in traffic capacity is nevertheless highly instructive as technical input to the capacity planning process. Furthermore, this kind of calculation requires the kind of cross-layer modelling we advocate in this paper.

7 Conclusion

One way to view the research we are carrying out is to ask the question: if a rack of machines in a datacenter was devoted to understanding as much as possible about what was happening, or what might happen, in the rest of the datacenter, how should one construct the software that runs on it?

The generic engine we envision in the long term maintains an accurate, cross-layer, online model of the state of a complete datacenter based on configuration data and real-time tracing, and supports detailed analytics queries to examine traffic flow, analyze root causes for failures, and aid in capacity and contingency planning.

The system we have described in this paper is very much an exploratory prototype, and we have only begun to explore what can be done with a facility for modeling and reasoning about all the layers of a datacenter stack.

Two, related directions lie ahead. The first is to push the kinds of analyses that can be performed with such a system. For example, our partners are particularly interested in capacity planning and worst-case failure identification, in the presence of complex cascading faults.

The second is how to build the system. To be able to run online at scale and deal with the complex, multi-paradigm queries we need is an interesting challenge in itself, integrating as it does relational, graph and stream processing. Recent work on unified query processing such as *Crackle* [11] is promising in this regard. A further requirement, however, is the ability to "fork" the model to simulate real-time, "what if" simulations.

In both cases, identifying a data center modelling engine as a separate component, reusable across different enterprises, is key to advancing the state of the art.

References

[1] Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D. A., and Zhang, M. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (New York, NY, USA, 2007), SIGCOMM '07, ACM, pp. 13–24.

[2] Barham, P., Donnelly, A., Isaacs, R., and Mortier, R. Using Magpie for request extraction and workload modelling. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6* (Berkeley, CA, USA, 2004), OSDI'04, USENIX Association, pp. 18–18.

[3] Barroso, L. A., and Hölzle, U. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. 2009.

[4] Chen, M. Y., Kiciman, E., Fratkin, E., Fox, A., and Brewer, E. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks* (Washington, DC, USA, 2002), DSN '02, IEEE Computer Society, pp. 595–604.

[5] Fonseca, R., Porter, G., Katz, R. H., Shenker, S., and Stoica, I. X-trace: A pervasive network tracing framework. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation* (Berkeley, CA, USA, 2007), NSDI'07, USENIX Association, pp. 20–20.

[6] Franklin, M. J. Making sense of big data with the Berkeley data analytics stack. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management* (New York, NY, USA, 2013), SSDBM, ACM, pp. 1:1–1:1.

[7] Kandula, S., Mahajan, R., Verkaik, P., Agarwal, S., Padhye, J., and Bahl, P. Detailed diagnosis in enterprise networks. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication* (New York, NY, USA, 2009), SIGCOMM '09, ACM, pp. 243–254.

[8] Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S., and Shanbhag, C. Dapper, a large-scale distributed systems tracing infrastructure. Tech. rep., Google, Inc., 2010.

[9] Troutwine, B. 10 billion a day, 100 milliseconds per: Monitoring real-time bidding at AdRoll. URL: `http://www.slideshare.net/BrianTroutwine1/10-billion-a-day-100-milliseconds-per-monitoring-realtime-bidding-at-adroll`. Slides of a talk given at Erlang Factory SF Bay Area, 6–7 March 2014.

[10] Wang, C., Kavulya, S., Tan, J., Hu, L., Kutare, M., Kasick, M. P., Schwan, K., Narasimhan, P., and Gandhi, R. Performance troubleshooting in data centers: an annotated bibliography? *Operating Systems Review 47*, 3 (2013), 50–62.

[11] Yoneki, E., and Roy, A. A unified graph query layer for multiple databases. Tech. Rep. UCAM-CL-TR-820, University of Cambridge, Computer Laboratory, Aug. 2012.