Cross-Examination of Datacenter Workload Modeling Techniques

Christina Delimitrou, Christos Kozyrakis
{cdel, kozyraki}@stanford.edu
Stanford University

Abstract
Datacenter workload modeling has become a necessity in recent years due to the emergence of large-scale applications and cloud data-stores, whose implementation remains largely unknown. Detailed knowledge of target workloads is critical in order to correctly provision performance, power and cost-optimized systems. In this work we aggregate previous work on datacenter workload modeling and perform a qualitative comparison based on the representativeness, accuracy and completeness of these designs. We categorize modeling techniques in two main approaches, in-breadth and in-depth, based on the way they address the modeling of the workload. The former models the behavior of a workload in specific system parts, while the latter traces a user request throughout its execution. Furthermore, we propose the early concept of a new design, which bridges the gap between these two approaches by combining some features from each one. Some first performance results based on this design appear promising as far as the accuracy of the model is concerned.

1 Introduction
The emergence of social networking and cloud data-stores has brought forward a challenge previously addressed with simple machine benchmarking. Datacenter workload modeling, is a critical requirement when evaluating different server configurations for performance, power and cost optimized large-scale systems. Inefficient server designs have a significant impact in the capital and operational cost of a datacenter (DC), since these inefficiencies get aggregated over several thousand servers. Therefore, a deep knowledge of target workloads is required. However, access to real applications’ code is rarely feasible, and replay of the entire application in different system configurations is impractical on a cost, time and reliability basis. Furthermore, conventional benchmarking is not applicable in large-scale DCs due to user behavior patterns that characterize large online services. This makes the necessity for accurate modeling of DC workloads even more urgent. A representative model that captures key features of large-scale applications can then be used to reproduce accurate workload profiles. One of the principal contributions of this methodology is the opportunities it offers in evaluating various system design challenges [13] without the need for access to real applications.

The necessity for a model is a common basis agreement. The ideal concept of workload modeling, however, has long been a matter of debate. Different approaches address the issue in fundamentally different ways, which further complexes the understanding of what a representative model should look like. In this work we perform an overview of previous work on workload modeling - specifically DC workloads - and present a qualitative comparison of their advantages and disadvantages. The main goal of this paper is to aggregate, categorize and put in perspective the work performed so far and provide insight on the direction that appears most profitable to adopt. Specifically, we differentiate the proposed designs in in-breadth and in-depth, based on the way they approach modeling. The former refers to modeling of specific system parts (e.g. storage or CPU modeling) while the latter proposes tracing a request through system layers. Both approaches have strong arguments in favor of their use and disadvantages that contend that on their own are insufficient to capture certain aspects of the workload. Furthermore, we present the early concept of a modeling approach that combines advantages of these two approaches. We address initial design issues and propose potential use cases for this work. The rest of this paper is organized as follows. Section 2 presents an overview of previous work on DC workload modeling while Section 3 discusses the advantages and disadvantages of each approach as far as the representativeness, practicality, and completeness of their models are concerned. Section 4 presents a new approach on workload modeling and Section 5 proposes potential use cases for this work. Finally, Section 6 addresses future work and concludes the paper.

2 Modeling Techniques Review
Workload modeling has attracted increased interest in recent years, due to the emergence of large-scale applications whose implementation remains largely unknown. The designs proposed vary as far as the aspects of the DC workload they capture, the granularity, ease-of-use and accuracy of their models [10]. There is, however, one major classification based on the design’s concept. The first approach (in-breadth) is more system-centric and targets modeling of specific system parts (e.g. storage, CPU, memory) in a manner that captures the features of the request in the corresponding subsystem. The second approach (in-depth) traces a user request through the system, in one or multiple tiers. The following subsections discuss the main trends in each approach as well as their most representative models.

2.1 In-Breadth Modeling
In-Breadth modeling targets modeling the behavior of a workload in specific parts of the system. Methods on modeling a workload in the storage, computation, memory and network subsystems have been proposed in a number of different ways and vary in the granularity and preciseness with which they capture the application’s features. We proceed to present an overview of the most relevant attempts of each type.
2.1.1 Storage Modeling

Storage accounts for a significant part of the DC’s operating and capital costs, therefore efficient design is critical for high quality systems. However, there has not been extensive work on modeling this part of the system, with most storage studies relying on trace-driven characterization [12, 14]. Traces are useful, but their value is limited by the system upon which they have been collected. A model allows for a far better understanding of a DC application’s storage behavior decoupled from the underlying platform.

Sankar et al. [1] propose a state diagram-based storage model that captures the I/O characteristics, as well as the spatial and temporal locality of storage requests. Extending this model to a hierarchical structure and implementing a tool that creates representative storage profiles, enables various storage system studies that would otherwise require access to application code or full application deployment.

Gulati et al. [24] propose characterization and modeling of storage I/O loads with applications to VM migration and application consolidation. For each workload they model the following I/O features: seek distance (i.e. randomness), I/O sizes, read:write ratio, and number of outstanding I/Os. Based on the profile of each workload they construct a model that predicts the expected latency to service I/O requests. They validate the accuracy of the model in predicting the impact different I/O load configurations have on latency.

Finally, Ozmen et al. [25] propose a workload model for storage profiles that does not depend on a trace-driven approach, to avoid the cost and platform dependency traces suffer from, but instead uses a Rome-based model [26]. Rome is a general purpose model intended to model storage workloads generated by any type of storage client. It perceives storage activity as a stream of stores characterized by parameters like: randomness, request rates, read/write mix, burstiness, and request size. This workload model is subsequently used to generate representative traces that span the entire database and maintain the mixture and frequency of different types of queries of the database workload.

2.1.2 CPU Modeling

Characterization of the computing requirements of workloads is a very frequently addressed problem. Abrahao et al. [9] propose a trace-based approach to characterize the utilization of CPU in a set of applications running on a shared cluster. They classify CPU Utilization patterns as periodic, noisy and spiky and resolve to PCA to process and categorize the large amount of trace data. They also recreate synthetic workloads with CPU utilization patterns that resemble those in the original application. Patwardhan et al. [4] take a step forward and focus on the CPU utilization patterns of web workloads, which resemble large scale DC applications. They perform a breakdown of CPU usage, focusing on the networking overhead, to identify the most likely cause of performance degradation in future online services and propose an analytical model that uses the application’s features to predict the throughput improvement due to protocol offload. They conclude that for dynamic applications, where most of the CPU time is dedicated to data processing, protocol offload will offer marginal benefits, while for static applications offloading the networking overhead to hardware can offer significant speedups.

Huang et al. [8] use workload characterization, with focus on CPU utilization, in order to apply DVFS during processor stalls due to long off-chip activities to improve energy efficiency. They derive a workload prediction model based on the application’s CPU utilization pattern to quantify the benefit from switching to a low power mode during e.g. batch I/O processes. Finally, Basaran et al. [18] use a fuzzy logic controller to model the CPU utilization for individual tasks, and achieve the required performance levels. They show that their model-free controller outperforms a PI and MPC controller even for workloads that vary dynamically.

Although characterization of CPU utilization patterns has significant prior work, deriving a CPU model remains relatively unexamined. The reason for this is the difficulty in decoupling CPU utilization from the underlying system platform. For a model to be useful, it should be valid for different system configurations without fundamental changes. However, when the metric used is utilization of computing resources, the model becomes instantly a reflection of the platform than the actual workload. In Section 4 we propose a differentiation in the way we perceive computing resources to address this issue.

2.1.3 Network Modeling

Network modeling is one of the most studied aspects of the system and recently has attracted increased interest due to the user behavior patterns that large scale applications experience. Feitelson [2] presents a detailed overview of methods used to characterize and model network request distributions. He proposes distribution fitting, through the Kolmogorov-Smirnov test, in order to identify the type of distribution of user-requests arrival. He also characterizes requests based on their stationarity, self-similarity, burstiness, and heavy tails, features present in most DC applications. Furthermore, he presents preliminary considerations on the correlation between task size, arrival rate and execution time, combining network and CPU modeling. His work presents very strong arguments towards the value of modeling for a variety of system studies.

As an extension to this work, Li [5] proposes detailed characterization of network and CPU intensive applications running on large scale grids. He analyses features like the job arrival rate, size, pseudoperiodicity, short and long-range dependence, as well as the correlation of these features with the execution time for a job. He proposes a two-phase approach to model these workload attributes. The first step consists of Model-Based Clustering in order to perform the distribution fitting. The second step generates autocorrelations that match the real data to create synthetic workloads. Evaluating the properties of these autocorrelations can provide a closer look to the performance impact request distributions have on large scale grids.

Joo et al. [19] propose network traffic modeling to resolve a set of performance-related problems. They compare two fundamentally different network models, an infinite-source-based model with no user variability that transfers a large but constant amount of data, and a SURGE-based [20] workload model, where traffic varies per user (i.e. number and rate of pages/objects/sizes requested from the server). They conclude that results for the two models vary greatly, therefore the accuracy of the model in capturing user behavior and the assumptions made for the network protocol are instrumental for the fidelity of the observed results.
Sengupta et al. [11] perform extensive characterization of request arrival rates for a series of OLTP workloads and propose an analytical model for distribution fitting and self-similarity recognition. They conclude that accurate modeling of network traffic, which most of the time diverges from the commonly-used Poisson distribution, can lead to improved decision making in performance/efficiency-related network studies.

Finally, we describe two papers that do not explicitly belong to network modeling but focus more on graph-driven techniques to derive user behavior patterns of large-scale applications. Luthi [27] proposes the use of multi-dimensional histograms (i.e., VU-lists) to model the job characteristics in Web applications. These histograms are collections of parameter vectors such as job arrival-rate distributions and job requirements of various system resources. Current applications of VU-lists include the analysis of closed queuing networks. Tang et al. [28] propose a framework that models long-term behavior of network activity by capturing the non-stationarity, burstiness and duration of requests. They also develop MediSyn, a publicly available synthetic streaming media workload generator, based on their model.

2.1.4 Memory Modeling

Memory utilization varies greatly among DC applications. For those with minimal locality, memory remains mostly underutilized, while for applications with hot and cold data, e.g., Search, memory becomes the bottleneck for performance, especially given the processor-memory speed gap. This has led to extensive characterization and modeling of memory access patterns for conventional and large scale applications. Barroso et al. [7] present a detailed characterization of memory utilization for a set of large scale OLTP, DSS and Web Search workloads. Using the information on memory access patterns they evaluate different memory trends and propose simple architectural optimizations in the memory system to improve performance.

Similarly, Jaleel [21] focuses on memory characterization of workloads from the SPEC CPU2000 and SPEC CPU2006 benchmark suites. He proposes the use of instrumentation-driven simulation based on Pin [22] to achieve fast, robust and simple characterization of the workloads’ memory requirements, with minimal overhead. He also evaluates the impact that various memory configuration parameters have on the workload’s performance.

Moro et al. [3] propose the use of the sequence of memory references (i.e., virtual page numbers) as a series of floating point numbers used to train an Ergodic Continuous Hidden Markov Model (ECHMM). They use this method to categorize the memory activity of each workload, and to generate synthetic traces. They prove their method to be significantly more accurate in determining the memory behavior of a workload than previously proposed methods.

2.2 In-Depth Modeling

On the other end of in-breadth modeling are techniques that trace requests as they advance through the system. In-depth modeling has the advantage of capturing the progress of a request’s execution, thus simulating a realistic application scenario. This Section describes some of the most representative modeling proposals based on this approach.

Liu et al. [23] propose an analytical model for 3-tier Web applications. Their model consists of three multi-station queueing models, which emulate the Web, Application and Database tier respectively. The requests in the first tier are generated using the TPC-W benchmark. The analytical model is proven to accurately predict the performance metrics (throughput and latency) of request servicing in the original application. A similar technique is proposed by Kamra et al. [33] where a 3-tier workload is simulated using a queuing model for admission control of HTTP requests using a PI controller.

Apart from simple queueing networks, layered queueing networks (LQNs) have also been proposed for the modeling of multiple-tier Web applications. Franks et al. [34] and Imielski [35] propose the use of LQNs in order to demonstrate the nested possession of multiple resources. The disadvantage of using LQNs for modeling of DC workloads is that, although they emulate the request arrival accurately, the complexity introduced in the model because of the multiple concurrent queues often makes it prohibitive for large scale experiments.

Ganapathi et al. [6] propose a statistics-based approach for cloud application modeling. Kernel Canonical Correlation Analysis (KCCA) is used to model and predict the execution time of MapReduce jobs based on task features. The eventual goal of this work is generating representative workloads that allow the evaluation of MapReduce optimizations. Meisner et al. [36] present SQS (Stochastic Queuing Simulation) for characterization and evaluation of DCs. SQS is based on queuing theory and statistical sampling to derive system models that scale well to thousands of machines. The first step of SQS consists of the characterization step. During this phase, empirical workload models are constructed in an online manner that records the task arrival rate and duration. From these empirical distributions, different workload profiles can be synthesized. In the second part of the algorithm these models, along with offline system models, are used to simulate the queuing network for different system configurations. The authors prove that SQS scales well, without significant overhead with appropriate tuning of the sampling parameters.

Sigelman et al. [29] and Ren et al. [30] present Dapper and GWP, two tools that compose Google’s tracing and profiling infrastructure respectively. The goal of Dapper is to provide detailed and localized information on performance bottlenecks of DC applications. This infrastructure has three fundamental design constraints: ubiquitous deployment on a very large scale, application-level transparency and very low computational overhead. The ubiquity of tracing stems from the need to continuously monitor the entire application to avoid hard to reproduce behaviors. Dapper shares some implementation features with Pinpoint [32] and Magpie [31], but also introduces data sampling to achieve low overhead and accomplishes very high application transparency, partially due to the homogeneity of the system’s infrastructure. Dapper starts tracing a request the moment it arrives in the front-end server and until the response is sent to the originating client, after having issued multiple RPCs to middle-tier and backend servers. In order for all messages sent and received from each server to be aggregated and associated with the request that initiated them, Dapper uses trees of nested RPCs, spans (i.e., tree nodes) and annotations. The
latter means that the tool relies on applications or middleware to tag all message records with a unique global identifier that ties each message to the originating request. By maintaining the timestamps between consecutive RPCs and sampling 1 out of 1000 requests, Dapper achieves complete in-depth modeling with marginal performance overhead (less than 1.5% in all cases).

GWP on the other hand, is a continuous profiling tool providing performance insights for cloud applications. Unlike Dapper, it operates at a higher level, sampling across machines, in order to identify trends in job scheduling and execution that enable optimizations at the software and hardware infrastructure. It collects information that includes high-level events like job arrival rate, and task sizes and low-level system information like CPU utilization, L1/L2 cache misses and branch misprediction percentages. GWP performs both whole-machine and per-process collection of profiles, which are then stored in GFS and linked to specific source-code. Finally, GWP implements adaptive per-application sampling to reduce the overhead of profile collecting while ensuring no critical information loss. GWP is currently used for hardware performance optimizations (e.g. processor microarchitecture, cache type and size).

Such tools are instrumental in emulating an application’s structure and identifying problematic points in its control flow, but, at the moment, lack the ability to model and recreate the characteristics of a workload apart from its network traffic, e.g. its storage, processor and memory activity.

3 Qualitative Comparison

In the previous Section we described various schemes for workload modeling and characterization. In this Section we perform a comparison between the two dominant approaches (in-breadth and in-depth) and discuss their main advantages and disadvantages.

3.1 In-Breadth Modeling

The main advantage of the in-breadth approach is its system-centric nature, which permits a better understanding of the workload’s behavior in specific parts of the system. Especially when the model captures multiple subsystems, this approach can reveal correlations between different aspects of the workload. Furthermore, since the model is tied to system parts, although the individual characteristics change between applications, its main structure remains constant. This offers a generic infrastructure across the application space, which simplifies understanding and extending the model. It also enables detailed examination of specific aspects of the application by fine-tuning the corresponding model. Finally, the fact that it relies on system-parameters facilitates the advance to a performance and power model for the DC, which enables more detailed system studies, than merely simulating the arrival-rate of user-requests. This correlation between workload and system model can prove invaluable when the eventual goal is large-scale simulation.

The most obvious disadvantage of this method is its inability to capture the time dependencies of a request as it progresses through the system. Not being able to capture an application’s structure, can result in invalid stressing of the system, which renders the model inaccurate. Therefore, on its own an in-breadth model can not consicely simulate an application with complicated control flow. Another potential disadvantage of this method is its complexity, which can become prohibitive for large scale experiments.

3.2 In-Depth Modeling

On the other end, the most important benefit from using an in-depth model is its ability to capture an application’s control flow, namely trace the steps of a request’s execution through the system. This property is invaluable when the target workloads are multi-tier applications with complicated structure. Furthermore, in-depth models focus on capturing the incoming request traffic. This results in very accurate user behavior models which are necessary to reproduce valid real-life scenarios. The simplicity of the model, which usually relies on a simple queueing network has made this technique appealing for large-scale experiments. Another advantage of in-depth modeling is the fact that it enables studies that require the notion of jobs or tasks. Studies that involve identifying performance bottlenecks for a specific job, performing error detection or sophisticated job mapping are only possible with an in-depth modeling scheme. There are, however, some disadvantages that prohibit in-depth modeling from addressing all the requirements of an accurate DC application model.

DC applications are not always easy to track, and features like replica management, sharding and back-up updates make their structure even more complicated. If the model is to capture all these inter-dependencies it can easily become intractable. Furthermore, since the entirety of the model depends on the application’s structure, each workload has a different model, which removes the benefit of having a common modeling infrastructure across the application space. Another disadvantage of this method is that, although accurate in capturing user behavior patterns, it does not capture the features of the workload in various subsystems. This impedes the derivation of a performance and/or power model for the system.

4 KOOZA: A Combined Approach

Previously we discussed the advantages and limitations of various proposed modeling approaches. The main drawback of the in-breadth approach is the absence of information on the application’s structure. On the other hand, the in-depth approach is oversimplified, only emulating the arrival-rate of user-requests, but not the request’s access features. In this section, we propose a new design, that bridges the gap between these approaches, by taking advantage of their corresponding benefits. KOOZA is a modular, primarily in-breadth approach with the ability to capture the application’s time dependencies. The model for each server comprises of four simple models that reflect the behavior of a workload, in the four main parts of the system: storage, processor, memory and network and a queue, configurable for each workload, that demonstrates the structure.
of the application, i.e. the order in which each model becomes active. For the storage, processor and memory we use Markov Chain Models, while for the network we use a simple queueing model to represent the arrival-rate of user requests. The storage and memory models reflect the type of requests (block size, type, randomness, inter-arrival times) and the spatial locality in the granularity of Logical Block Ranges (LBNs) and Memory Banks respectively. The processor model quantifies the CPU utilization achieved for a given request. The reason why Markov models are preferred for these three models is that we want to capture the sequence of states and the probabilities of switching between them. For the network model, the primary goal is to capture the arrival-rate of requests which makes a simple queueing model sufficient. In order to convey more detailed information on one or multiple aspects of the workload, the simple Markov Chain can be substituted by a corresponding hierarchical representation. Each one of the four models is trained using traces from the corresponding subsystem.

Figure 1 shows the structure of a popular DC application. For simplicity, we choose GFS [16], a large-scale file system, comprised only of third-tier servers. As a request comes from the network, it utilizes the CPU (and memory) to verify whether the data is available in the specific chunkserver and then performs a number of I/O operations in the storage system. Reversely, once the disk I/Os complete the CPU is being exercised in order to aggregate the data, before it is being transmitted over the network to the end user.

Figure 2 demonstrates the structure of the model for GFS. Each step of a request in Figure 1 is now replaced by the corresponding model of that part of the system. We note here that the four models are serialized only in the case of a single request and can obviously be concurrently active in a real application scenario. Scaling to multiple servers in order to simulate real-application scenarios requires multiple instances of the model.

In order to verify the validity of the model we perform some initial experiments. We use traces of simplified requests from GFS to train the models and verify that requests generated using the model have the same features and performance metrics as the original requests. We should note here that in these preliminary experiments we do not take into account the complete dataflow of GFS, but only examine simple GFS client - GFS chunkserver requests that comply with the structure of Figure 1. Although, real DC applications are a lot more complicated than that, these results provide some initial guarantees that the proposed model can indeed capture the features and performance metrics of large-scale applications. The experiments presented here are for one server, however we plan to extend this study to include more complicated scenarios as part of our future work. Table 2 shows these results for two examples of user requests. In both cases the synthetic request generated based on the model does not deviate from the original workload more than 1% for the request features and more than 6% for the performance metrics.

One of the advantages of this approach is that the basic structure of the model remains the same across different applications, providing a generalized infrastructure for a wide application space. At the same time, the queue describing the workload’s time correlations is application-dependent and adds a configurability to the model, based on each application’s structure. The detail of the model is configurable and since its structure is distributed (four simple models instead of a complicated one), the designer can adjust the level of detail to the part of the system that is of interest for each application or study. Such information on the workload’s behavior can prove to be instrumental in server configuration and DC provisioning which remain as some of the greatest challenges DC designers face today. Finally, given a unified address space in the DC, and since information on job/task ids is recorded the model can replicate effects like the TCP/IP incast problem, or other events involving multiple machines servicing the same request.

A potential disadvantage of this methodology is the complexity of training four models and extracting the time de-
pendencies for the application of interest. Training the four models requires collecting traces for the corresponding part of the system, a standard procedure for any DC configuration study, while creating the time-dependencies-queue requires tracing the complete round trip of a request through the system from issue to response. As far as the parameters required for each model, the modular structure of the design prohibits a high complexity. Furthermore, we can reduce the dimensionality of feature-space, to the ones necessary for a representative and succinct model, using techniques like PCA [17], SVD, sampling, or regression analysis. Additional detail increases the model’s complexity, and that remains a trade-off dependent on the application’s behavior and the study of interest. Table 1 summarizes a comparison between in-breadth, in-depth models and the model proposed here.

5 Applicability

The main incentive towards designing a representative model for DC applications is understanding their specific behavior towards designing a performance, power and cost optimized system for them. An obvious case of the opportunities this methodology offers is evaluating different server configurations without access to real DC application source-code. The storage model used in KOOZA has been effectively applied in storage system studies like SSD caching and defragmentation evaluation to improve performance and efficiency [37]. Apart from storage, the CPU and memory models can be used to evaluate different processor options, given the increased interest in small-core usage for energy efficiency in the DC [15], as well as the effects of a heterogeneous processor or memory system in Quality of Service (QoS) and TCO. Similarly, the network model can be used to assess the merit of different network topologies towards a performance or efficiency-optimized system. Even more interesting are the correlations that emerge between individual models. Studying these correlations can facilitate the development of a performance and power model for the datacenter, enabling system studies that would otherwise be impractical from a cost and time perspective. Although we acknowledge the level of complexity in this claim we believe that the insight a representative DC workload model offers can expedite the extraction of these models. Finally, detailed knowledge on the application can identify bottlenecks in performance and create incentive towards software optimizations (e.g., rewriting parts of the application or changing compiler/OS/middleware options).

6 Conclusions and Future Work

In this paper we have presented an overview of the most relevant previous work on workload modeling and a qualitative comparison between the two approaches that prevail among them. The in-breadth and in-depth techniques have been evaluated based on the representativeness, accuracy, ease-of-use and completeness of their designs and the most important advantages and disadvantages of each one have been presented. Furthermore, we presented the early concept of KOOZA, a new modeling technique, that combines advantages from in-breadth and in-depth methods by capturing both the time dependencies of a large-scale application and the features of user requests in individual subsystems. The main part of future work is implementing this design, and verifying the validity of its model. Some early results of synthetic workloads based on this model, presented here, create incentive towards a full implementation of KOOZA. This design appears promising is becoming a complete application modeling infrastructure for DC workloads with applications in server configuration, performance and power modeling and large-scale DC simulation.

References