

Answering “What-If” Deployment and Configuration Questions with WISE*

Mukarram Bin Tariq[‡], Amgad Zeitoun[§], Vytautas Valancius[‡], Nick Feamster[‡], Mostafa Ammar[‡]
mtariq@cc.gatech.edu, amgad@google.com, {valas,feamster,ammar}@cc.gatech.edu

[‡] School of Computer Science, Georgia Tech. Atlanta, GA [§] Google Inc. Mountain View, CA

Abstract

Designers of content distribution networks often need to determine how changes to infrastructure deployment and configuration affect service response times when they deploy a new data center, change ISP peering, or change the mapping of clients to servers. Today, the designers use coarse, back-of-the-envelope calculations, or costly field deployments; they need better ways to evaluate the effects of such hypothetical “what-if” questions before the actual deployments. This paper presents *What-If Scenario Evaluator (WISE)*, a tool that predicts the effects of possible configuration and deployment changes in content distribution networks. *WISE* makes three contributions: (1) an algorithm that uses traces from existing deployments to learn causality among factors that affect service response-time distributions; (2) an algorithm that uses the learned causal structure to estimate a dataset that is representative of the hypothetical scenario that a designer may wish to evaluate, and uses these datasets to predict future response-time distributions; (3) a scenario specification language that allows a network designer to easily express hypothetical deployment scenarios without being cognizant of the dependencies between variables that affect service response times. Our evaluation, both in a controlled setting and in a real-world field deployment at a large, global CDN, shows that *WISE* can quickly and accurately predict service response-time distributions for many practical *what-if* scenarios.

Categories and Subject Descriptors: C.2.3 [Computer Communication Networks]: Network Operations, Network Management

General Terms: Algorithms, Design, Management, Performance

Keywords: What-if Scenario Evaluation, Content Distribution Networks, Performance Modeling

1. INTRODUCTION

Content distribution networks (CDNs) for Web-based services comprise hundreds to thousands of distributed servers and data cen-

*This work is supported in part by NSF Awards CNS-0643974, CNS-0721581, and CNS-0721559.

[†]Work performed while the author was visiting Google Inc.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCOMM’08, August 17–22, 2008, Seattle, Washington, USA.
Copyright 2008 ACM 978-1-60558-175-0/08/08 ...\$5.00.

ters [1, 3, 9]. Operators of these networks continually strive to improve the response times for their services. To perform this task, they must be able to predict how service response-time distribution changes in various hypothetical *what-if* scenarios, such as changes to network conditions and deployments of new infrastructure. In many cases, they must also be able to reason about the *detailed* effects of these changes (e.g., what fraction of the users will see at least a 10% improvement in performance because of this change?), as opposed to just coarse-grained point estimates or averages.

Various factors on both short and long timescales affect a CDN’s service response time. On short timescales, response time can be affected by routing instability or changes in server load. Occasionally, the network operators may “drain” a data center for maintenance and divert the client requests to an alternative location. In the longer term, service providers may upgrade their existing facilities, move services to different facilities or deploy new data centers to address demands and application requirements, or change peering and customer relationships with neighboring ISPs. These instances require significant planning and investment; some of these decisions are hard to implement and even more difficult to reverse.

Unfortunately, reasoning about the effects of any of these changes is extremely challenging in practice. Content distribution networks are complex systems, and the response time perceived by a user can be affected by a variety of inter-dependent and correlated factors. Such factors are difficult to accurately model or reason about and back-of-the-envelope calculations are not precise.

This paper presents the design, implementation, and evaluation of *What-If Scenario Evaluator (WISE)*, a tool that estimates the effects of possible changes to network configuration and deployment scenarios on the service response time. *WISE* uses statistical learning techniques to provide a largely automated way of interpreting the *what-if* questions as statistical interventions. *WISE* takes as input packet traces from Web transactions to model factors that affect service response-time prediction. Using this model, *WISE* also transforms the existing datasets to produce a new datasets that are representative of the *what-if* scenarios and are also faithful to the working of the system, and finally uses these to estimate the system response time distribution.

Although function estimation using passive datasets is a common application in the field of machine learning, using these techniques is not straightforward because they can only predict the response-time distribution for a *what-if* scenario accurately if the estimated function receives an input distribution that is representative of the *what-if* scenario. Providing this input distribution presents difficulties at several levels, and is the key problem that *WISE* solves.

WISE tackles the following specific challenges. First, ***WISE must allow the network designers to easily specify what-if scenarios.*** A designer might specify a *what-if* scenario to change the

value of some network features relative to their values in an existing or “baseline” deployment. The designer may not know that such a change might also affect other features (or how the features are related). *WISE*’s interface shields the designers from this complexity. *WISE* provides a *scenario specification language* that allows network designers to succinctly specify hypothetical scenarios for arbitrary subsets of existing networks and to specify *what-if* values for different features. *WISE*’s specification language is simple: evaluating a hypothetical deployment of a new proxy server for a subset of users can be specified in only 2 to 3 lines of code.

Second, because the designer can specify a *what-if* scenario without being aware of these dependencies, ***WISE* must automatically produce an accurate dataset** that is both *representative* of the *what-if* scenario the designer specifies and *consistent* with the underlying dependencies. *WISE* uses a causal dependency discovery algorithm to discover the dependencies among variables and a statistical intervention evaluation technique to transform the observed dataset to a representative and consistent dataset. *WISE* then uses a non-parametric regression method to estimate the response time as a piece-wise smooth function for this dataset. We have used *WISE* to predict service response times in both controlled settings on the Emulab testbed and for Google’s global CDN for its Web-search service. Our evaluation shows that *WISE*’s predictions of response-time distribution are very accurate, yielding a median error between 8% and 11% for cross-validation with existing deployments and only 9% maximum cumulative distribution difference compared to ground-truth response time distribution for *what-if* scenarios on a real deployment as well as controlled experiments on Emulab.

Finally, ***WISE* must be fast**, so that it can be used for short-term and frequently arising questions. Because the methods relying on statistical inference are often computationally intensive, we have tailored *WISE* for parallel computation and implemented it using the Map-Reduce [16] framework, which allows us to process large datasets comprising hundreds of millions of records quickly and produce accurate predictions for response-time distributions.

The paper proceeds as follows. Section 2 describes the problem scope and motivation. Section 3 makes the case for using statistical learning for the problem of *what-if* scenario evaluation. Section 4 provides an overview of *WISE*, and Section 5 describes *WISE*’s algorithms in detail. We discuss the implementation in Section 6. In Section 7, we evaluate *WISE* for response-time estimation for existing deployments as well as for a *what-if* scenario based on a real operational event. In Section 8, we evaluate *WISE* for *what-if* scenarios for a small-scale network built on the Emulab testbed. In Section 9, we discuss various properties of the *WISE* system and how it relates to other areas in networking. We review related work in Section 10, and conclude in Section 11.

2. PROBLEM CONTEXT AND SCOPE

This section describes common *what-if* questions that the network designers pose when evaluating potential configuration or deployment changes to an existing content distribution network deployment.

Content Distribution Networks: Most CDNs conform to a two-tier architecture. The first tier comprises a set of globally distributed front-end (FE) servers that, depending on the specific implementation, provide caching, content assembly, pipelining, request redirection, and proxy functions. The second tier comprises backend (BE) servers that implement the application logic, and which might also be replicated and globally distributed. The FE and BE servers may belong to a single administrative entity (as is

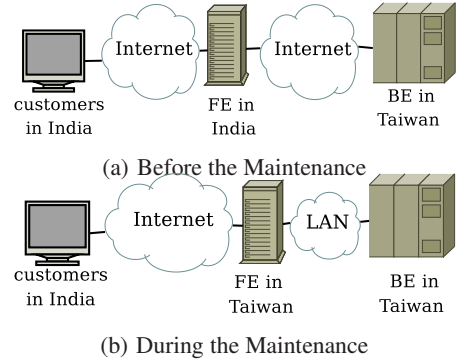


Figure 1: Network configuration for customers in India.

the case with Google [3]) or to different administrative entities, as with commercial content distribution networking service providers, such as Akamai [1]. The network path between the FE and BE servers may be over a public network or a private network, or a LAN when the two are co-located. CDNs typically use DNS redirection or URL-rewriting [13] to direct the users to the appropriate FE and BE servers; this redirection may be based on the user’s proximity, geography, availability, and relative server load.

An Example “What-if” Scenario: The network designers may want to ask a variety of *what-if* questions about the CDN configuration. For example, the network designers may want to determine the effects of deploying new FE or BE servers, changing the serving FE or BE servers for a subset of users, changing the size of typical responses, increasing capacity, or changing network connectivity, on the service response time. Following is a real *what-if* scenario from Google’s CDN for the Web-search service.

Figure 1 shows an example of a change in network deployment that could affect server response time. Google has an FE data center in India that serves users in India and surrounding regions. This FE data center uses BE servers located elsewhere in the world, including the ones located in Taiwan. On July 16, 2007, the FE data center in India was temporarily “drained” for maintenance reasons, and the traffic was diverted to a FE data center that is co-located with BE in Taiwan, resulting in a change in latency for the users in India. This change in the network configuration can be described as a *what-if* scenario in terms of change of the assigned FE, or more explicitly as changes in delays between FE and clients that occur due to the new configuration. *WISE* aims to predict the response-time distribution for reconfigurations before they are deployed in practice.

3. A CASE FOR MACHINE LEARNING

In this section, we present two aspects of *what-if* scenario evaluation that make the problem well-suited for machine learning: (1) an underlying model that is difficult to derive from first principles but provides a wealth of data; (2) a need to predict outcomes based on data that may not directly represent the desired *what-if* scenario.

The system is complex, but observable variables are driven by fundamental properties of the system. Unfortunately, in large complex distributed systems, such as CDNs, the parameters that govern the system performance, the relationships between these variables, as well as the functions that govern the response-time distribution of the system, are often complex and are characterized by randomness and variability that are difficult to model as simple readily evaluable formulas. Fortunately, the underlying fundamental properties and dependencies that determine a CDN’s

response time can be observed as correlations and joint probability distributions of the variables that define the system, including the service response time. By observing these joint distributions (e.g., response times observed under various conditions), machine learning algorithms can infer the underlying function that affects the response time. Because most production CDNs collect comprehensive datasets for their services as part of everyday operational and monitoring needs, the requisite datasets are typically readily available.

Obtaining datasets that directly represent the *what-if* scenario is challenging. Once the response-time function is learned, evaluating a *what-if* scenario requires providing this function with input data that is representative of the *what-if* scenario. Unfortunately, data collected from an existing network deployment only represents the current setup, and the system complexities make it difficult for a designer to manually “transform” the data to represent the new scenario. Fortunately, depending on the extent of the dataset that is collected and the nature of *what-if* scenario, machine learning algorithms can reveal the dependencies among the variables and use the dependency structure to intelligently *re-weigh and re-sample* the different parts of the existing dataset to perform this transformation. In particular, if the *what-if* scenario is expressed in terms of the changes to values of the variables that are observed in the dataset and the changed values or similar values of these variables are observed in the dataset even with small densities in the original dataset, then we can transform the original dataset to one that is representative of the *what-if* scenario as well as the underlying principles of the system, while requiring minimal input from the network designer.

4. WISE: HIGH-LEVEL DESIGN

WISE entails four steps: (1) identifying features in the dataset that affect response time; (2) constraining the inputs to “valid” scenarios based on existing dependencies; (3) specifying the *what-if* scenario; (4) estimating the response-time function and distribution. Each of these tasks raises a number of challenges, some of which are general problems with applying statistical learning in practice, and others are specific to *what-if* scenario evaluation. This section provides an overview and necessary background for these steps. Section 5 discuss the mechanisms in more depth; the technical report [24] provides additional details and background.

1. Identifying Relevant Features: The main input to *WISE* is a comprehensive dataset that covers many combinations of variables. Most CDNs have existing network monitoring infrastructure that can typically provide such a dataset. This dataset, however, may contain variables that are not relevant to the response-time function. *WISE* extracts the set of relevant variables from the dataset and discards the rest of the variables. *WISE* can also identify whether there are missing or latent variables that may hamper scenario evaluation (Sections 5.1 and 5.2 provide more details).

The nature of *what-if* scenarios that *WISE* can evaluate is limited by the input dataset—careful choice of variables that the monitoring infrastructure collects from a CDN can therefore enhance the utility of the dataset for evaluating *what-if* scenarios, choosing such variables is outside the scope of *WISE* system.

2. Preparing Dataset to Represent the What-if Scenario: Evaluating a *what-if* scenario requires values for input variables that “make sense.” Specifically, an accurate prediction of the response-time distribution for a *what-if* scenario requires a joint distribution of the input variables that is representative of the scenario and is also consistent with the dependencies that are inherent to the sys-

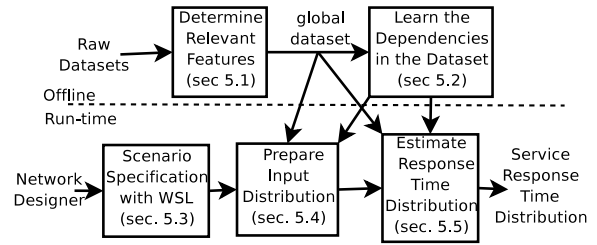
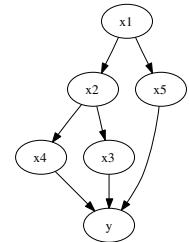


Figure 2: Main steps in the *WISE* approach.

tem itself. For instance, the distribution of the number of packets that are transmitted in the duration of a service session depends on the distribution of the size of content that the server returns in reply to a request; if the distribution of content size changes, then the distribution for the number of packets that are transmitted must also change in a way that is inherent to the system, e.g., the path-MTU might determine the number of packets. Further, the change might *cascade* to other variables that in turn depend on the number of packets. To enforce such consistency *WISE* learns the dependency structure among the variables and represents these relationships as a *Causal Bayesian Network (CBN)* [20]. We provide a brief background of CBN in this Section and explain the algorithm for learning the CBN in Section 5.2.

A CBN represents the variables in the dataset as a Directed Acyclic Graph (DAG). The nodes represent the variables and the edges indicate whether there are dependencies among the variables. A variable has a “causal” relationship with another variable, if a change in the value of the first variable *causes* a change in the values of the later. When conditioned on its parent variables, a variable x_i in a CBN is independent of all other variables in the DAG except its decedents; an optimal DAG for a dataset is one where we find the minimal parents for each node that satisfy the above property.

As an example of how the causal structure may facilitate scenario specification and evaluation, consider a dataset with five input variables ($x_1 \dots x_5$), and target variable y . Suppose that we discover a dependency structure among them as shown in the figure to the right. If *WISE* is presented with a *what-if* scenario that requires changes in the value of variable x_2 , then the distributions for variables x_1 and x_5 remains unchanged in the input distribution, and *WISE* needs to update only the distribution of the descendants of x_2 to maintain consistency. *WISE* constrains the input distribution by intelligently re-sampling and re-weighting the dataset using the causal structure as a guideline (see Section 5.4).



In general, correlation does not imply causation. Causal interpretation of association or correlation requires that the dataset is independent of the outcome variable (see the Counterfactual Model [20, 25]). A biased dataset can result in false or missing causal assertions; for example, we could falsely infer that a treatment is effective if, by coincidence, the dataset is such that more patients that are treated are healthy than the ones that are not treated. We can make the correct inference if we assign the patients randomly to the treatment because then the dataset would be independent of the outcome. Fortunately, because many computer networking phenomena are fundamentally similar throughout the Internet, we can assume that the datasets are unbiased. Still, frivolous relationships might arise; we address this further in Section 5.2.

3. Facilitating Scenario Specification: *WISE* presents the network designers with an easy-to-use interface in the form of a scenario specification language called *WISE-Scenario Language*

(WSL). The designers can typically specify the baseline setup as well as the hypothetical values for the scenario in 3-4 lines of WSL.

WSL allows the designers to evaluate a scenario for an arbitrary subset of customers. WSL also provides a useful set of built-in operators that facilitate scenario specification as relative changes to the existing values of variables or as new values from scratch. With WSL, the designers are completely shielded from the complexity of dependencies among the variables, because *WISE* automatically updates the dependent variables. We detail WSL and the process of scenario specification and evaluation in Sections 5.3 and 5.4.

4. Estimating Response-Time Distribution: Datasets for typical CDN deployments and *what-if* scenarios span a large multi-dimensional space. While non-parametric function estimation is a standard application in the machine learning literature, the computational requirements for accurately estimating a function spanning such a large space can be astronomical. To address this, *WISE* estimates the function in a piece-wise manner, and also structures the processing so that it is amenable to parallel processing. *WISE* also uses the dependency structure to reduce the number of variables that form the input to the regression function. Sections 5.5 and 5.6 provide more detail.

5. WISE SYSTEM

5.1 Feature Selection

Traditional machine-learning applications use various model selection criteria, e.g., Akaike Information Criterion (AIC), Mallows's C_p Test, or k-fold cross-validation [25], for determining appropriate subset of covariates for a learning problem. *WISE* forgoes the traditional model selection techniques in favor of simple pair-wise independence testing, because at times these techniques can ignore variables that might have interpretive value for the designer.

WISE uses simple pair-wise independence tests on all the variables in the dataset with the response-time variable, and discards all variables that it deems independent of the response-time variable. For each categorical variable (variables that do not have numeric meanings) in the dataset, such as, country of origin of a request, or AS number, *WISE* obtains the conditional distributions of response time for each categorical value, and discards the variable if all the conditional distributions of response time are statistically similar. To test this, we use Two-sample Kolmogorov-Smirnov (KS) goodness of fit test with a significance level of 10%.

For real-valued variables, *WISE* first tests for correlation with the response-time variable, and retains a variable if the correlation coefficient is greater than 10%. Unfortunately, for continuous variables, lack of correlation does not imply independence, so we cannot outright discard a variable if we observe small correlation. A typical example of such a variable in a dataset is the timestamp of the Web transaction, where the correlation may cancel out over a diurnal cycle. For such cases, we divide the range of the variable in question into small buckets and treat each bucket as a category. We then apply the same techniques as we do for the categorical variables to determine whether the variable is independent. There is still a possibility that we may discard a variable that is relevant, but this outcome is less likely if sufficiently small buckets are used. The bucket size depends on the variable in question; for instance, we use one-hour buckets for the time-stamp variable in the datasets.

5.2 Learning the Causal Structure

To learn the causal structure, *WISE* first learns the undirected graph and then uses a set of rules to orient the edges.

Learning the Undirected Graph: Recall that in a Causal Bayesian

```

1: WCD ( $\mathbf{V}$ ,  $\mathbf{W}_0$ ,  $\Delta$ )
   /*Notation
    $\mathbf{V}$ : set of all variables
    $\mathbf{W}_0$ : set of no-cause variables
    $\Delta$ : maximum allowable cardinality for separators
    $a \perp b$ : Variable  $a$  is independent of variable  $b$  */
2: Make a complete Graph on  $\mathbf{V}$ 
3: Remove all edges  $(a, b)$  if  $a \perp b$ 
4:  $\mathbf{W} = \mathbf{W}_0$ 
5: for  $c = 1$  to  $\Delta$  /*prune in the order of increasing cardinality*/
6:   LocalPrune ( $c$ )
7:   /*Try to separate neighbors of frontier variables  $\mathbf{W}'$ */
8:    $\forall w \in \mathbf{W}$ 
9:      $\forall z \in N(w)$  /*neighbors of  $w$ */
10:    if  $\exists \mathbf{x} \subseteq N(z) \setminus w : |\mathbf{x}| \leq c, z \perp w | \mathbf{x}$ 
11:    then /*found separator node(s)*/
12:       $S_{wz} = \mathbf{x}$  /*assign the separating nodes*/
13:      Remove the edge  $(w, z)$ 
14:      Remove edges  $(w', z)$ , for all the nodes  $w' \in \mathbf{W}'$ 
15:        that are also on path from  $w$  to nodes in  $\mathbf{W}_0$ 
16:    /*Update the new frontier variables*/
17:     $\mathbf{W} = \mathbf{W} \cup \mathbf{x}$ 

```

Figure 3: WISE Causal Discovery (WCD) algorithm.

Network (CBN), a variable, when conditioned on its parents, is independent of all other variables, except its descendants. Further an optimal CBN requires finding the smallest possible set of parents for each node that satisfy this condition. Thus by definition, variables a and b in the CBN have an edge between them, if and only if, there is a subset of *separating variables*, S_{ab} , such that a is independent of b given S_{ab} . This, in the general case, requires searching all the possible $O(2^n)$ combinations of the n variables in the dataset

WISE-Causal Discovery Algorithm (WCD) (Figure 3) uses a heuristic to guide the search of separating variables when we have prior knowledge of a subset of variables that are “not caused” by any other variables in the dataset, or that are determined by factors outside our system model (we refer to these variables as the *no-cause* variables). Further, WCD does not perform exhaustive search for separating variables, thus forgoing optimality for lower complexity.

WCD starts with a fully connected undirected graph on the variables and removes the edges among variables that are clearly independent. WCD then progressively finds separating nodes between a restricted set of variables (that we call *frontier* variables), and the rest of the variables in the dataset, in the order of increasing cardinality of allowable separating variables. Initially the frontier variables comprise only the no-cause variables. As WCD discovers separating variables, it adds them to the set of frontier variables.

The algorithm terminates when it has explored separation sets up to the maximum allowed cardinality $\Delta \leq n$, resulting in a worse case complexity of $O(2^\Delta)$. This termination condition means that certain variables that are separable are not separated: this does not result in false dependencies but potentially transitive dependencies may be considered direct dependencies. This sub-optimality does not affect the accuracy of the scenario datasets that *WISE* prepares, but it reduces the efficiency because it leaves the graph to be denser and the nodes having larger in-degree.

In the cases where the set of no-cause variables is unknown, *WISE* relies on the PC-algorithm [23], which also performs search

for separating nodes in the order of increasing cardinality among all pair of variables, but not using the frontier variables.

Orienting the Edges: *WISE* orients the edges and attempts to detect latent variables using the following simple rules, well known in the literature; we reproduce the rules here for convenience and refer the reader to [20] for further details.

1. Add outgoing edges from the no-cause variables.
2. If node c has nonadjacent neighbors a and b , and $c \in S_{ab}$, then orient edges $a \rightarrow c \leftarrow b$ (unmarked edges).
3. For all nonadjacent nodes, a, b with a common neighbor c , if there is an edge from a to c , but not from b to c , then add a marked edge $c \xrightarrow{*} b$.
4. If a and b are adjacent and there is directed path of only marked edges from a to b , then add $a \rightarrow b$

In the resulting graph, any unmarked, bi-directed, or undirected edges signify possible latent variables and *ambiguity in causal structure*. In particular, $a \rightarrow b$ means either a really causes b or there is a common latent cause L causing both a and b . Similarly, $a \leftrightarrow b$, signifies a definite common latent cause, and undirected edge between a and b implies either a causes b , b causes a , or a common latent cause in the underlying model.

Addressing False Causal Relationships: False or missing causal relationships can occur if the population in the dataset is not independent of the outcome variables. Unfortunately, because *WISE* relies on passive datasets this is a fundamental limitation that cannot be avoided. However, we expect that because the basic principles of computer networks are similar across the Internet, and the service providers use essentially the same versions of software throughout their networks, the bias in the dataset that would significantly affect the causal interpretation is not common. If such biases do exist, they will likely be among datasets from different geographical deployment regions. To catch such biases, we recommend using a training dataset with *WISE* that is obtained from different geographical locations. We can infer causal structure for each geographical region separately; if the learned structure is different, the differences must be carefully examined in light of the knowledge of systems internal working.

Lastly, while *WISE* depends on the CBN for preparing the scenario dataset, it is not necessary that the CBN is learned automatically from the dataset; the CBN can be supplied, entirely, or in part by a designer who is well-versed with the system.

5.3 Specifying the “What-If” Scenarios

Figure 4 shows the grammar for *WISE*-Specification Language (WSL). A scenario specification with WSL comprises a *use*-statement, followed by optional scenario *update*-statements.

The *use*-statement specifies a condition that describes the subset of present network for which the designer is interested in evaluating the scenario. This statement provides a powerful interface to the designer for choosing the baseline scenario: depending on the features available in the dataset, the designer can specify a subset of network based on location of clients (such as country, network address, or AS number), the location of servers, properties of service sessions, or a combination of these attributes.

The *update*-statements allow the designer to specify *what-if* values for various variables for the service session properties. Each scenario statement begins with either the INTERVENE, or the ASSUME keyword and allows conditional modification of exactly one variable in the dataset.

When the statement begins with the INTERVENE keyword, *WISE* first updates the value of the variable in question. *WISE* then uses the causal dependency structure to make the dataset *consistent*

```

scenario = use_stmt {update_stmt};
use_stmt = "USE" ("*" | condition_stmt)<EOL>;
update_stmt = ("ASSUME"|"INTERVENE") (set_directive |
    setdist_directive) [condition_stmt]<EOL>;
set_directive = "SET" ["RADIAL"* | "FIXED"]
    var set_op value;
setdist_directive = "SETDIST" feature
    dist_name([param])| "FILE" filename);
condition_clause = "WHERE" condition;
condition = simple_cond | compound_cond;
simple_cond = compare_clause | (simple_cond);
compound_cond = (simple_cond ("AND"|"OR")
    (simple_cond|compound_cond));
compare_clause = (var rel_op value) | member_test;
member_test = feature "IN" (value {,value});
set_op = "+=" | "-=" | "*=" | "\=" | "=";
rel_op = "<=" | ">=" | "<" | "=>" | "<" | ">";
var = a variable from the dataset;

```

Figure 4: Grammar for *WISE* Specification Language (WSL).

with the underlying dependencies. For this *WISE* uses a process called *Statistical Intervention Effect Evaluation* (Section 5.4).

Advanced designers can override the intelligent update behavior by using the ASSUME keyword in the update statement. In this case *WISE* updates the distribution of the variable specified in the statement but does not attempt to ensure that the distribution of the dependent variables are correspondingly updated. *WISE* allows this functionality for cases where the designers believe that the scenario that they wish to evaluate involves changes to the underlying invariant laws that govern the system. Examples of scenario specification with WSL will follow in Section 7.

5.4 Preparing Representative Distribution for the “What-If” Scenarios

This section describes how *WISE* uses the dataset, the causal structure, and the scenario specification from the designer to prepare a meaningful dataset for the *what-if* scenario.

WISE first filters the global dataset for the entries that match the conditions specified in the *use-statement* of the scenario specification to create the *baseline* dataset. *WISE* then executes the update-statements, one statement at a time, to change the baseline dataset. To ensure consistency among variables after every INTERVENE update statement, *WISE* employs a process called *Statistical Intervention Effect Evaluation*; the process is described below.

Let us denote the action requested on a variable x_i in the *update*-statement as $\text{set}(x_i)$. We refer to x_i as the *intervened* variable. Let us also denote the set of variables that are children of x_i in the CBN for the dataset as $\mathcal{C}(x_i)$. Then the statistical intervention effect evaluation process states that the new distribution of children of x_i is given as: $\Pr\{\mathcal{C}(x_i)|\text{set}(x_i)\}$. The intuition is that because the parent node in a CBN has a causal effect on its descendent nodes, we expect that a change in the value of the parent variable must cause a change in the value of the children. Further, the new distribution of children variables would be one that we would expect to observe under the *changed* values of the parent variable.

To apply this process, *WISE* conditions the *global* dataset on the new value of the intervened variable, $\text{set}(x_i)$, and the existing values of the all the other parents of the children of the intervened variable, $\mathcal{P}(\mathcal{C}(x_i))$, in the baseline dataset to obtain an empirical distribution. *WISE* then assigns the children a random value from this distribution. *WISE* thus obtains a subset of the global dataset in which the distribution of $\mathcal{C}(x_i)$ is consistent with the action $\text{set}(x_i)$ as well as the underlying dependencies.

Because the causal effect cascades to all the decedents of x_i , *WISE* repeats this process recursively, considering $\mathcal{C}(x_i)$ as the in-

tervened variables and updating the distributions of $\mathcal{C}(\mathcal{C}(\mathbf{x}_i))$, and so on, until all the descendants of \mathbf{x}_i (except the target variable) are updated. *WISE* cannot update the distribution of a descendant of \mathbf{x}_i until the distribution of all of its ancestors that are descendant of \mathbf{x}_i has been updated. *WISE* thus carefully orders the sequence of the updates by traversing the CBN DAG breadth-first, beginning at node \mathbf{x}_i .

WISE sequentially repeats this process for each statement in the scenario specification. The updated dataset produced after each statement serves as the input dataset for the next statement. Once all the statements are executed, the dataset is the representative joint distribution variables for the entire *what-if* scenario.

When the causal structure has ambiguities, *WISE* proceeds as follows. When the edge between two variables is undirected, *WISE* maintains the consistency by always updating the distribution of one if the distribution of the other is updated. For latent variables case, *WISE* assumes an imaginary variable, with directed edges to variables a and b and uses the resulting structure to traverse the graph while preparing the input distribution.

5.5 Estimating Response Time Distribution

Finding the new distribution of response time is also a case of intervention effect evaluation process. We use a non-parametric regression method to estimate the expected response-time distribution, instead of assigning a random value from the constrained empirical distribution as in the previous section, because the designers are interested in the expected values of the response time for each request. In particular, we use a standard Kernel Regression (KR) method, with a radial basis Kernel function (see [24, 26] for details) to estimate the response time for each request in the dataset. To address the computational complexity, *WISE* applies the KR in a piece-wise manner; the details follow in the next section.

5.6 Addressing the Computational Scalability

Because CDNs are complex systems, the response time may depend on a large number of variables, and the dataset might comprise hundreds of millions of requests spanning a large multi-dimensional space. To efficiently evaluate the *what-if* scenarios, *WISE* must address how to efficiently organize and utilize the dataset. In this section, we discuss our approach to these problems.

1. Curse of Dimensionality: As the number of dimensions (variables in the dataset) grow, exponentially more data is needed for similar accuracy of estimation. *WISE* uses the CBN to mitigate this problem. In particular, because when conditioned on its parents, a variable is independent of all variables except its descendants, we can use only the parents of the target variable in the regression function. Because the cardinality of the parent-set would typically be less than the total number of variables in the dataset, the accuracy of the regression function is significantly improved for a given amount of data. Due to this, *WISE* can afford to use fewer training data points with the regression function and still get good accuracy. Also, because the time complexity for the KR method is $\mathcal{O}(kn^3)$, with k variables and n points in the training dataset, *WISE*'s technique results in significant computational speedup.

2. Overfitting: The density of the dataset from a real deployment can be highly irregular; usually there are many points for combinations of variable values that represent the *normal* network operation, while the density of dataset is sparser for combinations that represent the fringe cases. Unfortunately, because the underlying principle of most regression techniques is to find parameters that minimize the errors on the training data, we can end up with parameters that minimize the error for high density regions of the dataset

but give poor results in the fringes—this problem is called overfitting. The usual solution to this problem is introducing a smoothness penalty in the objective function of the regression method, but finding the right penalty function requires cross-validation, which is usually at least quadratic in the size of the *global* dataset¹. In the case of CDNs, even one day of data may contain entries for millions of requests, which makes the quadratic complexity of these algorithms inherently unscalable.

WISE uses piece-wise regression to address this problem. *WISE* divides the dataset into small pieces, that we refer to as *tiles* and performs regression independently for each tile. *WISE* further prunes the *global* dataset to produce a training dataset so that the density of training points is more or less even across all the tiles.

To decompose the dataset, *WISE* uses fixed-size buckets for each dimension in the dataset for most of the variable value space. If the bucket sizes are sufficiently small, having more data points beyond a certain threshold does not contribute appreciably to the response-time prediction accuracy. With this in mind, *WISE* uses two thresholds n_{min} and n_{max} and proceeds with tile boundaries as follows. *WISE* decides on a bucket width b_i along each dimension i , and forms boundaries in the space of the dataset at integer multiples of b_i along each dimension; for categorical variables, *WISE* uses each category as a separate bucket. For each tile, *WISE* obtains a uniform random subset of n_{max} points that belong in the tile boundaries from the global dataset and adds them to the training dataset. If the dataset has fewer than n_{min} data points for the tile, the tile boundaries are folded to merge it with neighboring tile. This process is repeated until the tile has n_{min} number of points. Ultimately, most of the tiles have regular boundaries, but for some tiles, especially those on the fringes, the boundaries can be irregular. Once the preparation of training data is complete, we use cross-validation to derive regression parameters for each tile; the complexity is now only $\mathcal{O}(n_{max}^2)$ for each tile.

3. Retrieval of Data: With large datasets, even the mundane tasks, such as retrieving training and test data during the input distribution preparation and response-time estimation phases are challenging. Quick data retrieval and processing is imperative here because both of these stages are *online*, in the sense that they are evaluated when the designer specifies the scenario.

WISE expedites this process by intelligently indexing the training data off-line and the test data as it is created. Tiles are used here as well: Each tile is assigned a *tile-id*, which is simply a string formed by concatenating the tile's boundaries in each dimension. All the data points that lie in the tile boundaries are assigned the *tile-id* as a key that is used for indexing. For the data preparation stage, *WISE* performs the *tile-id* assignment and indexing along the dimensions comprising the parents of most commonly used variables, and for the regression phase, the tiling and indexing is performed for the dimensions comprising the parents of the target variable. Because the *tile-ids* use fixed length bins for most of the space, mapping of a point to its tile can be performed in constant time for most of the data-points using simple arithmetic operations.

4. Parallelization and Batching: We have carefully designed the various stages in *WISE* to support parallelization and batching of jobs that use similar or same data. In the training data preparation stage, each entry in the dataset can be independently assigned its *tile-id* based key because *WISE* uses regular sized tiles. Similarly, the regression parameters for each tile can be learned independently

¹Techniques such as in [10] can reduce the complexity for such N-body problems but are still quite complex than the approximations that *WISE* uses.

and in parallel. In the input data preparation stage, *WISE* batches the test and training data that belong in a tile and fetch the data from the training data for all of these together. Finally, because *WISE* uses piece-wise regression to evaluate the effects of intervention, it can batch the test and training data for each tile; further because the piece-wise computation are independent, they can take place in parallel.

6. IMPLEMENTATION

We have implemented *WISE* with the Map-Reduce framework [16] using the Sawzall logs processing language [22] and Python Map-Reduce libraries. We chose this framework to best exploit the parallelization and batching opportunities offered by the *WISE* design². We have also implemented a fully-functional prototype for *WISE* using a Python front-end and a MySQL backend that can be used for small scale datasets. We provide a brief overview of the Map-Reduce based implementation here.

Most steps in *WISE* are implemented using a combination of one or more of the four Map-Reduce patterns shown in Figure 5. *WISE* uses *filter* pattern to obtain conditional subsets of dataset for various stages. *WISE* uses the *Tile-id Assignment* pattern for preparing the training data. We set the n_{min} and n_{max} thresholds to 20 and 50, respectively to achieve 2-5% confidence intervals. In the input data preparation phase, the *use*-statement is implemented using the *filter* pattern. The *update*-statements use *update* pattern for applying the new values to the variable in the statement. If the *update*-statement uses the INTERVENE keyword then *WISE* uses the *Training & Test Data Collation* pattern to bring together the relevant test and training data and update the distribution of the test data in a batched manner. Each *update*-statement is immediately followed by the *Tile-id Assignment* pattern because the changes in the value of the data may necessitate re-assignment of the tile-id. Finally, *WISE* uses the *Training & Test Data Collation* pattern for piece-wise regression. Our Map-Reduce based implementation can evaluate typical scenarios in about 5 minutes on a cluster of 50 PCs while using nearly 500 GB of training data.

7. EVALUATING WISE FOR A REAL CDN

In this section, we describe our experience applying *WISE* to a large dataset obtained from Google’s global CDN for Web-search service. We start by briefly describing the CDN and the service architecture. We also describe the dataset from this CDN and the causal structure discovered using WCD. We also evaluate *WISE*’s ability to predict response-time distribution for the *what-if* scenarios.

7.1 Web-Search Service Architecture

Figure 6(a) shows Google’s Web-search service architecture. The service comprises a system of globally distributed HTTP reverse proxies, referred to as Front End (FE) and a system of globally distributed clusters that house the Web servers and other core services (the Back End, or BE). A DNS based request redirection system redirects the user’s queries to one of the FEs in the CDN. The FE process forwards the queries to the BE servers, which generate dynamic content based on the query. The FE caches static portions of typical reply, and starts transmitting that part to the requesting user as it waits for reply from the BE. Once the BE replies, the dynamic content is also transmitted to the user. The FE servers may or may not be co-located in the same data center with the BE

²Hadoop [11] provides an open-source Map-Reduce library. Modern data-warehousing appliances, such the ones by Netezza [18], can also exploit the parallelization in *WISE* design.

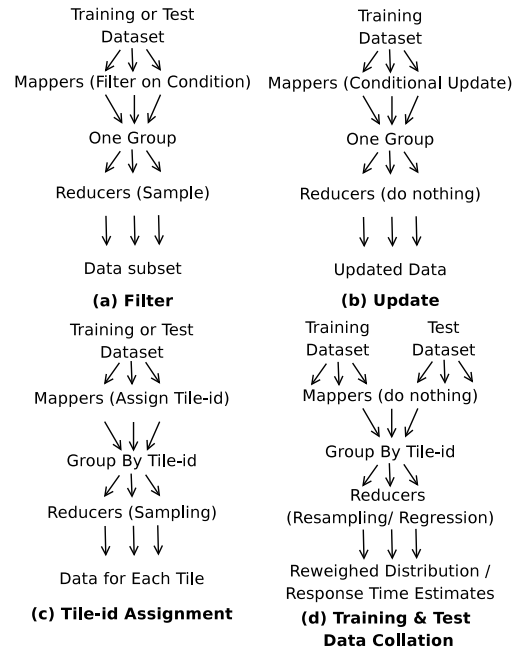


Figure 5: Map-Reduce patterns used in *WISE* implementation.

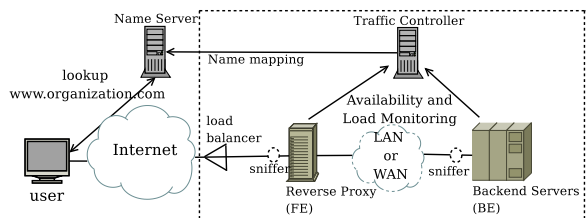
servers. If they are co-located, they can be considered to be on the same local area network and the round-trip latency between them is only a few milliseconds. Otherwise, the connectivity between the FE and the BE is typically on a well-provisioned connection on the public Internet. In this case the latency between the FE and BE can be several hundred milliseconds.

The *server response time* for a request is the time between the instance when the user issues the HTTP request and the instance when the last byte of the response is received by the users. We estimate this value as the sum of the round-trip time estimate obtained from the TCP three-way handshake, and the time between the instance when the request is received at the FE and when the last byte of the response is sent by the FE to user. The key contributors to server response time are: (i) the transfer latency of the request from the user to the FE (ii) the transfer latency of request to the BE and the transfer latency of sending the response from the BE to the FE; (iii) processing time at the BE, (iv) TCP transfer latency of the response from the FE to the client; and (v) any latency induced by loss and retransmission of TCP segments.

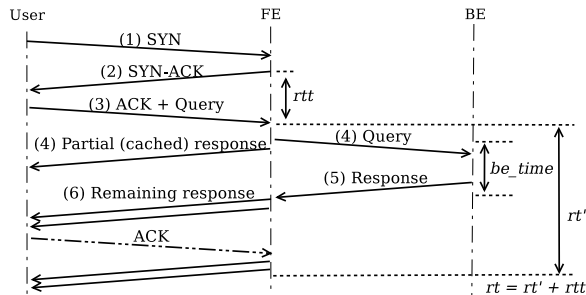
Figure 6(b) shows the process by which a user’s Web search query is serviced. This message exchange has three features that affect service response time in subtle ways, making it hard to make accurate “back-of-the-envelope calculations” in the general case:

1. Asynchronous transfer of content to the user. Once the TCP handshake is complete, user’s browser sends an HTTP request containing the query to the FE. While the FE waits on a reply from the BE, it sends some static content to the user; this content—essentially a “head start” on the transfer—is typically brief and constitutes only a couple of IP packets. Once the FE receives the response from the BE, it sends the response to the client and completes the request. A client may use the same TCP connection for subsequent HTTP requests.

2. Spliced TCP connections. FE processes maintain several TCP connections with the BE servers and reuse these connections for forwarding user requests to the BE. FE also supports HTTP pipelin-



(a) Google's Web-search Service Architecture



(b) Message Exchange

Figure 6: Google's Web-search service architecture and message exchange for a request on a fresh TCP connection.

ing, allowing the user to have multiple pending HTTP requests on the same TCP connection.

3. Spurious retransmissions and timeouts. Because most Web requests are short TCP transfers, the duration of the connection is not sufficient to estimate a good value for the TCP retransmit timer and many Web servers use default values for retransmits, or estimate the timeout value from the initial TCP handshake round-trip time. This causes spurious retransmits for users with slow access links and high serialization delays for MTU sized packets.

7.2 Data

We use data from an existing network monitoring infrastructure in Google's network. Each FE cluster has network-level sniffers, located between the FE and the load-balancers, that capture traffic and export streams in *tcpdump* format. A similar monitoring infrastructure captures traffic in the BE. Although the FE and BE servers use NTP for time synchronization, it is difficult to collate the traces from the two locations using only the timestamps. Instead, we use the hash of each client's IP, port and part of query along with the timestamp to collate the request between the FE and the BE. *WISE* then applies the relevance tests (ref. Sec. 5.1) on the features in the dataset collected in this manner. Table 1 describes the variables that *WISE* found to be relevant to the service response-time variable.

7.3 Causal Structure in the Dataset

To obtain the causal structure, we use a small sampled data subset collected on June 19, 2007, from several data center locations. This dataset has roughly 25 million requests, from clients in 12,877 unique ASes.

We seed the WCD algorithm with the *region* and *ts* variables as the *no-cause* variables. Figure 7 shows the causal structure that WCD produces. Most of the causal relationships in Figure 7 are straightforward and make intuitive sense in the context of networking, but a few relationships are quite surprising. WCD detects a relationship between the *region* and *sB* attribute (the size of the result page); we found that this relationship exists due to the differences in the sizes of search response pages in different languages

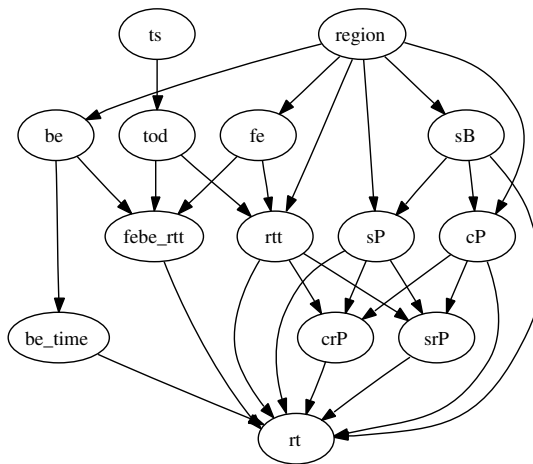


Figure 7: Inferred causal structure in the dataset. $A \rightarrow B$ means A causes B .

and regions. Another unexpected relationship is between *region*, *cP* and *sP* attributes; we found that this relationship exists due to different MTU sizes in different parts of the world. Our dataset, unfortunately, did not have load, utilization, or data center capacity variables that could have allowed us to model the *be_time* variable. All we observed was that the *be_time* distribution varied somewhat among the data centers. Overall, we find that WCD algorithm not only discovers relationships that are faithful to how networks operate but also discovers relationships that might escape trained network engineers.

Crucially, note that many variables are not direct children of the *region*, *ts*, *fe* or *be* variables. This means that when conditioned on the respective parents, these variables are independent of the region, time, choice of FE and BE, and we can use training data from past, different regions, and different FE and BE data centers to estimate the distributions for these features! Further, while most of the variables in the dataset are correlated, the in-degree for each variable is smaller than the total number of variables. This reduces the number of dimensions that *WISE* must consider for estimating the value of the variables during scenario evaluation, allowing *WISE* to produce accurate estimates, more quickly and with less data.

7.4 Response-Time Estimation Accuracy

Our primary metric for evaluation is *prediction accuracy*. There are two sources of error in response-time prediction: (i) error in response-time estimation function (Section 5.5) and (ii) inaccurate input, or error in estimating a valid input distribution that is representative of the scenario (Section 5.4). To isolate these errors, we first evaluate the estimation accuracy alone and later consider the overall accuracy for a complete scenario in Section 7.5.

To evaluate accuracy of the piece-wise regression method in isolation we can try to evaluate a scenario: "What-if I make no changes to the network?" This scenario is easy to specify with WSL by not including any optional scenario update statements. For example, a scenario specification with the following line: `USE WHERE country==deu` would produce an input distribution for the response-time estimation function that is representative of users in Germany without any error and any inaccuracies that arise would be due to regression method. To demonstrate the prediction accuracy, we present results for three such scenarios:

- (a) `USE WHERE country==deu`
- (b) `USE WHERE country==zaf`

Feature	Description
ts, tod	A time-stamp of instance of arrival of the request at the FE. We also extract the hourly time-of-day (tod) from the timestamp.
sB	Number of bytes sent to the user from the server; this does not include any data that might be retransmitted or TCP/IP header bytes.
sP	Number of packets sent by the server to the client excluding retransmissions.
cP	Number of packets sent by the client that are received at the server, excluding retransmissions.
srP	Number of packets retransmitted by the server to the client, either due to loss, reordering, or timeouts at the server.
crP	Number of packets retransmitted by the client that are received at the server.
region	We map the IP address of the client to a region identifier at the country and state granularity. We also determine the /24(s24) and /16(s16) network addresses and the originating AS number. We collectively refer to these attributes as region.
fe, be	Alphanumeric identifiers for the FE data center at which the request was received and the BE data center that served the request.
rtt	Round-trip time between the user and FE estimated from the initial TCP three-way handshake.
febe_rtt	The network level round-trip time between the front end and the backend clusters.
be_time	Time between the instance that BE receives the request forwarded by the FE and when the BE sends the response to the FE.
rt	The response time for the request as seen by the FE (see Section 7.1). Response time is also our target variable.

Table 1: Features in the dataset from Google’s CDN.

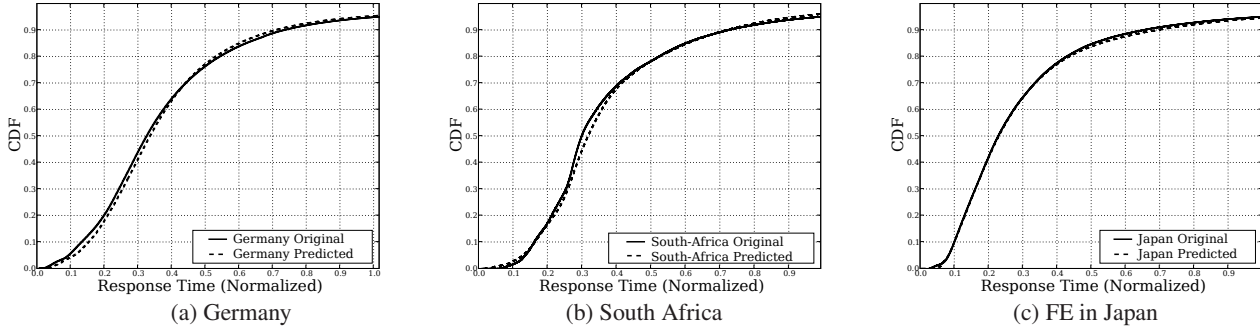


Figure 8: Prediction accuracy: comparison of normalized response-time distributions for the scenarios in Section 7.4.

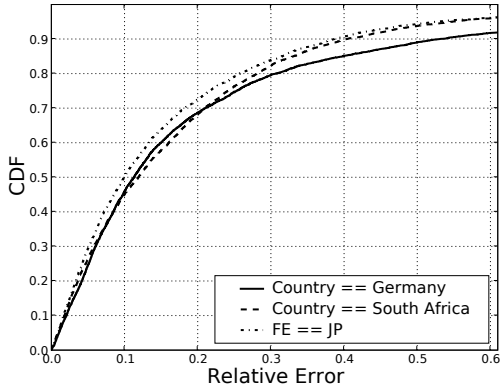


Figure 9: Relative prediction error for the scenarios in Figure 8.

(c) USE WHERE fe==jp

The first two scenarios specify estimating the response-time distribution for the users in Germany, and South Africa, respectively (deu and zaf are the country codes for these countries in our dataset), and the third scenario specifies estimating the response-time distribution for users that are served from FE in Japan (jp is the code for the FE data center in Japan); this data center primarily serves users in South and East Asia. We used the dataset from the week of June 17-23, 2007 as our training dataset and predicted the response time field for dataset for the week of June 24-30, 2007.

Figures 8(a), (b), and (c), show the results for the three scenarios, respectively. The ground-truth distribution for response time is based on the response-time values observed by the monitoring infrastructure for June 24-30 period, and is shown in solid line. The response time for the same period that *WISE* predicts is shown in a dotted line. Further, in Figure 9 we present the relative prediction

Dataset	<i>WISE</i>	AKM	CSA
deu	18%	55%	45%
zaf	12%	35%	120%
jp	15%	38%	50%

Table 2: Comparison of median relative error for response-time estimation for scenarios in Section 7.4.

error for these experiments. The error is defined as $|rt - \hat{rt}|/rt$, where rt is the ground-truth value and \hat{rt} is the value that *WISE* predicts. The median error lies between 8-11%.

Comparison with TCP Transfer Latency Estimation: It is reasonable to ask how well we could do using a simpler parametric model. To answer this question, we relate the problem of response-time estimation to that of estimating TCP-transfer latency, because Google uses TCP to transfer the data to clients. Many parametric models for TCP transfer latency are known; we have implemented two recent models by Arlitt [2] and Cardwell [5] for comparison. We refer to these methods as AKM and CSA, respectively. We modified the two methods to account for the additional latency that occurs due to the round-trip from the FE to the BE, and the time that the backend servers take (*be_time*). AKM uses a compensation multiplier called ‘*CompWeight*’ to minimize the error for each trace. We computed the value of this weight that yielded minimum error. For the CSA scheme, we estimated the loss rate as the ratio of number of retransmitted packets and the total number of packets sent by the server in each of the three scenarios. We used the values of other model parameters that are correct to the best of our knowledge for each trace.

Table 2 presents the median error for requests that had at least one retransmitted packet in the response ($srP > 1$). The three datasets had 3.0%, 57.0%, and 9.1% sessions with at least one retransmit, and the loss rates (calculated as described above) are

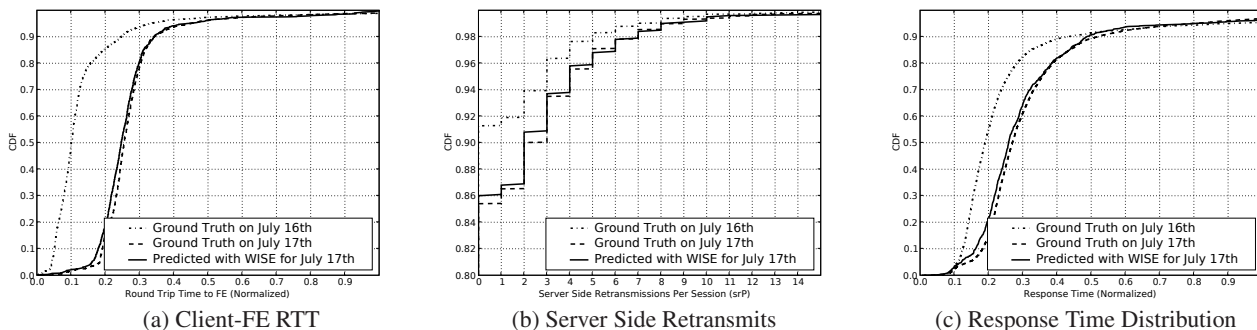


Figure 10: Predicted distribution for response time and intermediary variables for the India data center drain scenario.

0.9%, 1.2%, 22.5%, and 3.2%, respectively. While the AKM method does not account for retransmits and losses, the compensation factor improves its accuracy. The CSA method models TCP more faithfully, but it is not accurate for short TCP transfers. The estimation error with *WISE* is at least 2.5-2.9 times better than the other scheme across different traces without using any trace specific compensation.

7.5 Evaluating a Live What-if Scenario

We evaluate how *WISE* predicts the response-time distribution for the affected set of customers during the scenario that we presented in Section 2 as a motivating example. In particular, we will focus on the effect of this event on customers of AS 9498, which is a large consumer ISP in India.

To appreciate the complexity of this scenario, consider what happens on the ground during this reconfiguration. First, because the FE in Taiwan is co-located with the BE, *febe_rtt* reduces to a typical intra-data center round-trip latency of 3ms. Also we observed that the average latency to the FE for the customers of AS 9498 increased by about 135ms as they were served from FE in Taiwan (*tw*) instead of the FE in India (*im*).

If the training dataset already contains the *rtt* estimates for customers in AS 9498 to the *fe* in Taiwan then we can write the scenario in two lines as following:

```
USE WHERE as_num==9498 AND fe==im AND be==tw
INTERVENE SET fe=tw
```

WISE uses the CBN to automatically update the scenario distribution. Because, *fe* variable is changed, *WISE* updates the distribution of children of *fe* variable, which in this case include *febe_rtt* and *rtt* variables. This in-turn causes a change in children of *rtt* variable, and similarly, the change cascades down to the *rt* variable in the DAG. In the case when such *rtt* is not included in the training dataset, the value can be explicitly provided as following:

```
USE WHERE as_num==9498 AND fe==im AND be==tw
INTERVENE SET febe_rtt=3
INTERVENE SET rtt+=135
```

We evaluated the scenario using the former specification. Figure 10 shows ground truth response-time distribution as well as distributions for intermediary variables (Figure 7) for users in AS 9498 between hours of 12 a.m. and 8 p.m. on July 16th and the same hours on July 17th, as well as the distribution estimated with *WISE* for July 17th for these variables. We observe only slight underestimation of the distributions with *WISE*—this underestimation primarily arises due to insufficient training data to evaluate the variable distribution for the peripheries of the input distribution; *WISE* was not able to predict the response time for roughly 2% of the requests in the input distribution. Overall, maximum cumulative

distribution differences³ for the distributions of the three variables were between 7-9%.

8. CONTROLLED EXPERIMENTS

Because evaluating *WISE* on a live production network is limited by the nature of available datasets and variety of events with which we can corroborate, we have created a small-scale Web service environment using the Emulab testbed [7]. The environment comprises a host running the Apache Web server as the backend or BE, a host running the Squid Web proxy server as the frontend or FE, and a host running a multi-threaded wget Web client that issues request at an exponentially distributed inter-arrival time of 50 ms. The setup also includes delay nodes that use dummynet to control latency.

To emulate realistic conditions, we use a one-day trace from several data centers in Google’s CDN that serve users in the USA. We configured the resource size distribution on the BE, as well as to emulate the wide area round-trip time on the delay node based on this trace.

For each experiment we collect *tcpdump* data and process it to extract a feature set similar to one described in Table 1. We do not specifically emulate losses or retransmits because these occurred for fewer than 1% of requests in the USA trace. We have conducted two *what-if* scenario experiments in this environment; these are described below.

Experiment 1: Changing the Resource Size: For this experiment, we used just the backend server and the client machine, and used the delay node to emulate wide area network delays. We initially collected data using the resource size distribution based on the real trace for about two hours, and used this dataset as the training dataset. For the *what-if* scenario, we replaced all the resources on the server with resources that are half the size, and collected the test dataset for another two hours. We evaluated the test case with *WISE* using the following specification:

```
USE *
INTERVENE SET FIXED sB/=2
```

Figure 11(b) presents the response-time distribution for the original page size (dashed), the observed response-time distribution with halved page sizes (dotted), and the response-time distribution for the *what-if* scenario predicted with *WISE* using the original page size based dataset as input (solid). The maximum CDF distance in this case is only 4.7%, which occurs around the 40th percentile.

³We could not use the relative error metric here because the requests in the input distribution prepared with *WISE* for *what-if* scenario cannot be pair-wise matched with ones in the ground-truth distribution; maximum distribution difference is a common metric used in statistical tests, such as Kolmogorov-Smirnov Goodness-of-Fit Test.

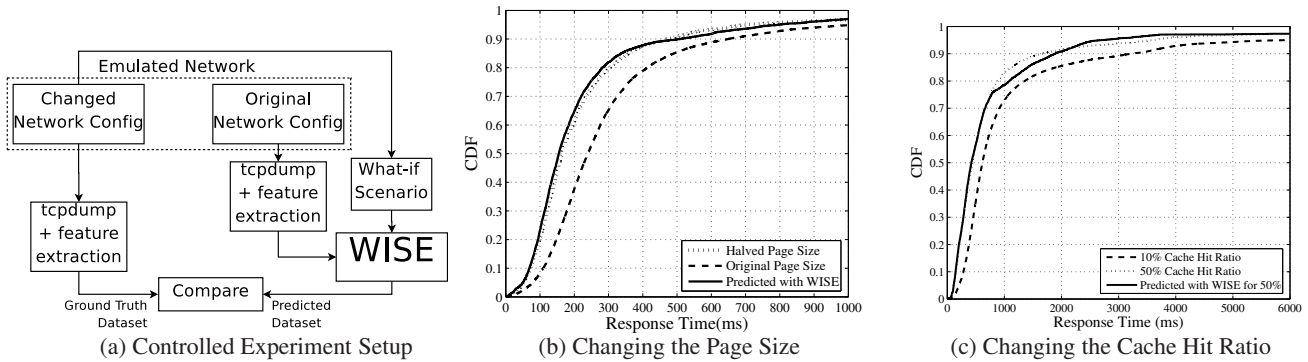


Figure 11: Controlled *what-if* scenarios on Emulab testbed: experiment setup and results.

Experiment 2: Changing the Cache Hit Ratio: For this experiment, we introduced a host running a Squid proxy server to the network and configured the proxy to cache 10% of the resources uniformly at random. There is a delay node between the client and the proxy as well as the proxy and the backend server, each emulates trace driven latency as in the previous experiment. For the *what-if* scenario, we configured the Squid proxy to cache 50% of the resources uniformly at random. To evaluate this scenario, we include binary variable b_cached for each entry in the dataset that indicates whether the request was served by the caching proxy server or not. We use about 3 hours of trace with 10% caching as the training dataset, and use *WISE* to predict the response-time distribution for the case with 50% caching by using the following specification:

```
USE *
INTERVENE SETDIST b_cached FILE 50pcdist.txt
```

The `SETDIST` directive tells *WISE* to update the b_cached variable by randomly drawing from the empirical distribution specified in the file, which in this case contains 50% 1s and 50% 0s. Consequently, we intervene 50% of the requests to have a cached response.

Figure 11(c) shows the response-time distribution for the 10% cache-hit ratio (dashed), the response-time distribution with 50% cache-hit ratio (dotted), and the response-time distribution for the 50% caching predicted with *WISE* using the original 10% cache-hit ratio based dataset as input (solid). *WISE* predicts the response time quite well for up to the 80th percentile, but there is some deviation for the higher percentiles. This occurred because the training dataset did not contain sufficient data for some of the very large resources or large network delays. The maximum CDF distance in this case is 4.9%, which occurs around 79th percentile.

9. DISCUSSION

In this section, we discuss the limitations of and extensions to *WISE*. First we discuss what can and what cannot be predicted with *WISE*. We also describe issues related to parametric and non-parametric techniques. Lastly we discuss how the framework can be extended to other realms in networking.

What Can and Cannot Be Predicted: The class of *what-if* scenarios that can be evaluated with *WISE* depends entirely on the dataset that is available; in particular, *WISE* has two requirements:

First, *WISE* requires expressing the *what-if* scenario in terms of (1) variables in the dataset and (2) manipulation of those variables. At times, it is possible for dataset to capture the effect of the variable without capturing the variable itself. In such cases, *WISE* cannot evaluate any scenarios that require manipulation of that hidden variable. For example, the dataset from Google, presented earlier,

does not include the TCP timeout variable even though this variable has an effect on response time. Consequently, *WISE* cannot evaluate a scenario that manipulates the TCP timeout.

Second, *WISE* also requires that the dataset contains values of variables that are similar to the values that represent the *what-if* scenario. If the global dataset does not have sufficient points in the space where the manipulated values of the variables lie, the prediction accuracy is affected, and *WISE* raises warnings during scenario evaluation.

WISE also makes stability assumptions, i.e., the causal dependencies remain unchanged under any values of intervention, and the underlying behavior of the system that determines the response times does not change. We believe that this assumption is reasonable as long as the fundamental protocols and methods that are used in the network do not change.

Parametric vs. Non-Parametric: *WISE* uses the assumption of functional dependency among variables to update the values for the variables during the statistical intervention evaluation process. In the present implementation, *WISE* only relies on non-parametric techniques for estimating this function but nothing in the *WISE* framework prevents using parametric functions. If the dependencies among some or all of the variables are parametric or deterministic, then we can improve *WISE*'s utility. Such a situation can in some cases allow *extrapolation* to predict variable values outside of what has been observed in the training dataset.

What-if Scenarios in other Realms of Networking: We believe that our work on evaluating *what-if* scenarios can be extended to incorporate other realms, such as routing, policy decisions, and security configurations by augmenting the reasoning systems with a decision evaluation system, such as *WISE*.

Our ultimate goal is to evaluate *what-if* scenarios for high-level goals, such as, “What if I deploy a new server at location X?”, or better yet, “How should I configure my network to achieve certain goal?”; we believe that *WISE* is an important step in this direction.

10. RELATED WORK

We are unaware of any technique that uses *WISE*'s approach of answering *what-if* deployment questions, but *WISE* is similar to previous work on TCP throughput prediction and the application of Bayesian inference to networking problems.

A key component in response time for Web requests is TCP transfer latency. There has been significant work on TCP throughput and latency prediction using TCP modeling [2,5,19]. Due to inherent complexity of TCP these models make simplifying assumptions to keep the analysis tractable; these assumptions may produce inaccurate results. Recently, there has been effort to embrace

the complexity and using past behavior to predict TCP throughput. He et al. [12] evaluate predictability using short-term history, and Mirza et al. [17] use machine-learning techniques to estimate TCP throughput — these techniques tend to be more accurate. We also use machine-learning and statistical inference in our work, but techniques of [17] are not directly applicable because they rely on estimating path properties immediately before making a prediction. Further, they do not provide a framework for evaluating *what-if* scenarios. The parametric techniques, as we show in Section 7.4, unfortunately are not very accurate for predicting response-time.

A recent body of work has explored use of Bayesian inference for fault and root-cause diagnosis. SCORE [15] uses spatial correlation and shared risk group techniques to find the best possible explanation for observed faults in the network. Shrink [14] extends this model to a probabilistic setting, because the dependencies among the nodes may not be deterministic due to incomplete information or noisy measurements. Sherlock [4] additionally finds causes for poor performance and also models fail-over and load-balancing dependencies. Rish et al. [21] combine dependency graphs with active probing for fault-diagnosis. None of these work, however, address evaluating *what-if* scenarios for networks.

11. CONCLUSION

Network designers must routinely answer questions about how specific deployment scenarios affect the response time of a service. Without a rigorous method for evaluating such scenarios, the network designers must rely on ad hoc methods or resort to costly field deployments to test their ideas. This paper has presented *WISE*, a tool for specifying and accurately evaluating *what-if* deployment scenarios for content distribution networks. To our knowledge, *WISE* is the first tool to automatically derive causal relationships from Web traces and apply statistical intervention to predict networked service performance. Our evaluation demonstrates that *WISE* is both fast and accurate: it can predict response time distributions in “what if” scenarios to within a 11% error margin. *WISE* is also easy to use: its scenario specification language makes it easy to specify complex configurations in just a few lines of code.

In the future, we plan to use similar techniques to explore how causal inference can help network designers better understand the dependencies that transcend beyond just performance related issues in their networks. *WISE* represents an interesting point in the design space because it leverages almost no domain knowledge to derive causal dependencies; perhaps *what-if* scenario evaluators in other domains that rely almost exclusively on domain knowledge (e.g., [8]) could also leverage statistical techniques to improve accuracy and efficiency.

Acknowledgments

We would like to thank Andre Broido and Ben Helsley at Google, and anonymous reviewers for the valuable feedback that helped improve several aspects of our work. We would also like to thank Jeff Mogul for sharing source code for the methods in [2].

12. REFERENCES

- [1] Akamai Technologies. www.akamai.com
- [2] M. Arlitt, B. Krishnamurthy, J. Mogul. Predicting short-transfer latency from TCP arcana: A trace-based validation. *IMC* 2005.
- [3] L.A. Barroso, J. Dean, U. Holzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*. Vol. 23, No. 2. pp 22–28
- [4] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. Maltz, M. Zhang. Towards Highly Reliable Enterprise Network Services via Inference of Multi-level Dependencies. *ACM SIGCOMM* 2007.
- [5] N. Cardwell, S. Savage, T. Anderson. Modeling TCP Latency. *IEEE Infocomm* 2000.
- [6] G. Cooper. A Simple Constraint-Based Algorithm for Efficiently Mining Observational Databases for Causal Relationships. *Data Mining and Knowledge Discovery* 1, 203-224. 1997.
- [7] Emulab Network Testbed. <http://www.emulab.net>
- [8] N. Feamster and J. Rexford. Network-Wide Prediction of BGP Routes. *IEEE/ACM Transactions on Networking*. Vol. 15. pp. 253–266
- [9] M. Freedman, E. Freudenthal, D. Mazieres. Democratizing Content Publication with Coral. *USENIX NSDI* 2004.
- [10] A. Gray, A. Moore, ‘N -Body’ Problems in Statistical Learning. *Advances in Neural Information Processing Systems* 13. 2000.
- [11] Lucene Hadoop. <http://lucene.apache.org/hadoop/>
- [12] Q. He, C. Dovrolis, M. Ammar. On the Predictability of Large Transfer TCP Throughput. *ACM SIGCOMM* 2006.
- [13] A. Barbir, et al. Known Content Network Request Routing Mechanisms. *IETF RFC* 3568. July 2003.
- [14] S. Kandula, D. Katabi, J. Vasseur. Shrink: A Tool for Failure Diagnosis in IP Networks. *MineNet Workshop SIGCOMM* 2005.
- [15] R. Kompella, J. Yates, A. Greenberg, A. Snoeren. IP Fault Localization Via Risk Modeling. *USENIX NSDI* 2005.
- [16] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *USENIX OSDI* 2004.
- [17] M. Mirza, J. Sommers, P. Barford, X. Zhu. A Machine Learning Approach to TCP Throughput Prediction. *ACM SIGMETRICS* 2007.
- [18] Netezza <http://www.netezza.com/>
- [19] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. *IEEE/ACM Transactions on Networking*. Vol 8. pp. 135-145
- [20] J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press. 2003.
- [21] I. Rish, M. Brodie, S. Ma. Efficient Fault Diagnosis Using Probing. *AAAI Spring Symposium on DMDP*. 2002.
- [22] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. *Interpreting the Data: Parallel Analysis with Sawzall*. *Scientific Programming Journal*. Vol. 13. pp. 227–298.
- [23] P. Sprites, C. Glymour. An Algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review* 9. *USENIX Symposium on Internet Technologies and Systems*. 1997.
- [24] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, M. Ammar. Answering “What-if” Deployment and Configuration Questions with WISE. *Georgia Tech Technical Report GT-CS-08-02*. February 2008.
- [25] L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer Texts in Statistics. 2003.
- [26] J. Wolberg. *Data Analysis Using the Method of Least Squares*. Springer. Feb 2006.