# Fuzzy Prophet: Parameter Exploration in Uncertain Enterprise Scenarios

Oliver Kennedy[*], Steve Lee[†], Charles Loboz[†], Slawek Smyl[†], Suman Nath[‡]
[*]EPFL & Cornell, [†]Microsoft Corporation, [‡]Microsoft Research
oliver.kennedy@epfl.ch, {stevlee, cloboz, slsmyl, sumann}@microsoft.com

## ABSTRACT

We present Fuzzy Prophet, a probabilistic database tool for constructing, simulating and analyzing business scenarios with uncertain data. Fuzzy Prophet takes externally defined probability distribution (so called VG-Functions) and a declarative description of a target scenario, and performs Monte Carlo simulation to compute probability distribution of the scenario's outcomes. In addition, Fuzzy Prophet supports parameter optimization,where probabilistic models are parameterized and a large parameter space must be explored to find parameters that optimize or achieve a desired goal. Fuzzy Prophet's key innovation is to use *fingerprints* that can identify parameter values producing correlated outputs of a user-provided stochastic function and to reuse computations across such values. Fingerprints significantly expedite the process of parameter exploration in offline optimization and interactive what-if exploration tasks.

## Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Database Applications—*Scientific databases, Statistical databases*

## General Terms

Design, Performance

## Keywords

Probabilistic database, Monte Carlo, Black box, Simulation

## 1. INTRODUCTION

There is an increasing need for enterprises to evaluate various business scenarios to assess and manage their financial, engineering, and operational risks arising from uncertain data. For example, as part of an enterprise analytics team at Microsoft Windows Azure cloud platform, we often need to simulate a large range of permutations of a scenario in order to identify which business decisions will optimize certain goal or lead to a desired outcome. For example, consider the task of deciding when to purchase new hardware in a datacenter. For this, we need to construct a scenario from various predictive models for resource (e.g., CPU core) demand and availability. By repeatedly evaluating the scenario with different purchase dates, we can identify a purchase date that provides a desirable tradeoff between maintenance costs and the risk of running out of capacity. Scenarios of such nature are not specific to Windows Azure and occur in many enterprises. However, due to inherent uncertainty in input models, simulating and evaluating such complex business scenarios can be extremely challenging without effective tools.

At Microsoft, we have developed the Fuzzy Prophet system, which aims to streamline the process of designing and simulating business scenarios on uncertain models. Fuzzy Prophet is a tool for constructing parameterized business scenarios and interactively exploring the effects of varying different parameters. Its workflow consists of three stages: (1) Analysts use specialized tools like R to model individual characteristics of a system. (2) Individual models are declaratively combined into a full business scenario, describing different system characteristics and the consequences of their interaction (e.g., demand for and availability of CPU cores, and the consequent risk of not having enough). (3) Fuzzy Prophet helps users identify a set of parameter values that optimize or achieve desired goals specified either by a constrained optimization problem, or by interacting with the user.

The first two steps in Fuzzy Prophet's workflow are ideally addressed by probabilistic database (PDB) systems[1, 2, 4, 5, 7, 8]. In a probabilistic database, probability distributions may be instantiated, stored, and queried using (nearly) standard SQL semantics. Several PDB systems, MCDB [4] and PIP [5] in particular, allow users to specify arbitrary probability distributions by using stochastic black-box variable generation functions (typically referred to as VG-Functions). Fuzzy Prophet is built as a wrapper around simplified PDB functionality, loosely based on MCDB, and built into Microsoft SQL-Server. VG-Functions enable the use of specialized tools for model instantiation, while SQL links baseline models into progressively more complex models, and entire business scenarios; hierarchically specifying the scenario allows each tier of the model to be thoroughly and independently validated.

Repeatedly evaluating a VG-Function, only making small changes to its parameter values can be wasteful; the function's output distributions for similar parameter values might be correlated. However, forcing users to specify the (potentially) complex interactions between a VG-Function and its parameters negates the generality and simplicity of defining models through VG-Functions.

The primary contribution of Fuzzy Prophet is a novel *fingerprinting* technique (discussed in greater depth in [6]) that identifies correlations between executions of a VG-Function with different parameter values. When simulating the outcome of two different parameterizations of the same scenario, these correlations allow us to re-map the simulation's output from one parameterization to the

[*]Work performed while interning at Microsoft Research


*SIGMOD'11,* June 12–16, 2011, Athens, Greece.
ACM 978-1-4503-0661-4/11/06.

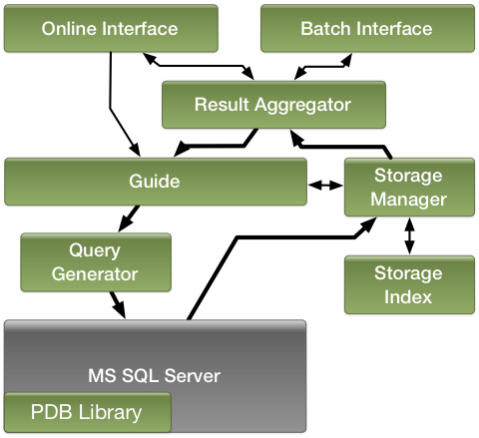


**Figure 1: The architectural design of Fuzzy Prophet.**

other and reduce the work associated with re-evaluating different permutations of the scenario.

Fuzzy Prophet can be used in an offline or offline mode. In offline mode, it identifies parameter values that optimize or achieve a desired goal. Execution is expedited by using fingerprints to avoid redundant computation. In online mode, users are presented with a live, progressively refined view of the effects of changing particular parameters. While interacting with the user, Fuzzy Prophet maintains a set of *basis distributions* containing the output of prior scenario evaluation runs. When evaluating the scenario with a new set of parameter values, Fuzzy Prophet first attempts to correlate the scenario's output distribution for one set of parameters to one or more basis distributions by matching their fingerprints, resulting in a lower time to first-accurate-guess.

## 2. PROPHET ARCHITECTURE

Users interact with Fuzzy Prophet through one of two interfaces: an online interface for user-directed parameter exploration, and an offline processing interface for automated parameter optimization. Users specify business scenarios as queries using Transact SQL (TSQL) with Fuzzy Prophet's PDB extensions and several pieces of metadata. The business scenario specification includes the definitions of parameters, which appear in the query as ordinary TSQL variables. The specification also includes metadata that either expresses a set of constrained optimization goals for the offline mode, or visualization directives for the online mode. For an example, see Section 3.

**Scenario Evaluation.** In both modes, Fuzzy Prophet operates on a cycle shown in Figure 1: (1) The *Guide* component directs scenario evaluation by producing a sequence of *instances*, each representing a concrete valuation for each parameter and model variable in the scenario (a possible world in PDB terminology). (2) The sequence of instances is batched and accepted by a *Query Generator*, which produces a pure TSQL query. The query is processed on a standard MS SQL Server install. (3) The results of the TSQL query are passed to a *Storage Manager*, which manages the set of basis distributions. Query results are used for fingerprinting purposes (as described below). (4) The *Result Aggregator* produces expectations, standard deviations, and other desired metrics. If Fuzzy Prophet is being used in online mode, these metrics are displayed to the user directly. In either case, the results are fed back to the Guide to direct its sampling strategy.

**Fingerprinting.** The core of Fuzzy Prophet is its fingerprinting technique. The fingerprint of a VG-Function is a concise and easily-computable data structure that summarizes its output distribution. Thus, a fingerprint can be used to efficiently determine a function's correlation with another function, or its own instantiations under different parameter values. After such correlation has been detected, Jigsaw can avoid expensive Monte Carlo estimation (and the associated VG-Function invocations) for a target point in the parameter space by using outputs for an already-explored, correlated point. Moreover, when a simulation is Markovian (where the simulation consists of a series of steps, each depending on the simulation's output for the prior step), outputs of successive steps often remain strongly correlated. This is particularly true for many processes of interest that are built around discontinuities, with discrete events occurring at random points in time (e.g., the nondeterministic date when new hardware comes online). Fingerprints can identify such *Markovian dependencies*, enabling automated generation of simple non-Markovian estimators. These estimators, valid for regions of the Markov chain, allow Fuzzy Prophet to skip the corresponding portions of the simulation.

The specific fingerprinting technique we use in this paper is based loosely on random testing [3], a well known technique in software engineering: the fingerprint of a parameterized stochastic function is simply a sequence of its outputs under a fixed sequence of random inputs (i.e., seed of its pseudorandom number generator). The use of a fixed set of random seeds ensures a deterministic relationship between correlated outputs of the stochastic functions.

Fingerprinting is discussed further in [6].

## 3. DEMONSTRATION

We demonstrate Prophet using a business scenario described in Section 3.1. Our demonstration consists of two components. First, users are introduced to the online Fuzzy Prophet interface, consisting of a set of sliders for setting parameter values, and a graph showing the per-week demand, capacity, and chance of overload (Section 3.2). Second, Fuzzy Prophet is run in offline mode on the same query and we will visualize how Prophet avoids redundant computation by exploiting fingerprints (Section 3.3).

### 3.1 Risk vs Cost of Ownership

This demo explores a specific commonly occurring business scenario: a Fuzzy Prophet end-user wishes to know when to purchase new server hardware for their cloud service. Delaying the purchases will reduce the overall cost of ownership, but increase the chance of running out of capacity within the cloud.

This scenario, presented in Figure 2, is constructed out of two input models forecasting the availability and demand for CPU cores in a Windows Azure datacenter[1].

The scenario employs four parameters: `@current`, `@purchase1`, `@purchase2`, and `@feature`. Each parameter is specified in terms of a discrete set of values, each corresponding to one week out of the coming year. The first parameter indicates the "current" week being simulated, the second two indicate the week of each of two anticipated hardware purchases, and the third indicates the week during which a particular software feature is released.

This scenario employs two models: `DemandModel` and `CapacityModel`, expressed using TSQL Table Generating functions. The `DemandModel` is a daily demand forecast expressed as a simple gaussian. A second gaussian is added to the first after the

[1]This demonstration uses models representative of those that prompted the development of Fuzzy Prophet. Data used in our simulations is arbitrarily chosen for intellectual property reasons.

```
-- DEFINITION --
DECLARE PARAMETER @current AS RANGE 0 TO 52 STEP BY 1;
DECLARE PARAMETER @purchase1 AS RANGE 0 TO 52 STEP BY 4;
DECLARE PARAMETER @purchase2 AS RANGE 0 TO 52 STEP BY 4;
DECLARE PARAMETER @feature AS SET (12,36,44);
SELECT DemandModel(@current, @feature)
          AS demand,
       CapacityModel(@current, @purchase1, @purchase2)
          AS capacity,
       CASE WHEN capacity < demand THEN 1 ELSE 0 END
          AS overload
  INTO results;
-- ONLINE MODE --
GRAPH OVER @current
      EXPECT overload WITH bold red,
      EXPECT capacity WITH blue y2,
      EXPECT_STDDEV demand WITH orange y2;
-- OFFLINE MODE --
OPTIMIZE SELECT @feature, @purchase1, @purchase2
         FROM results
         WHERE MAX(EXPECT overload) < 0.01
         GROUP BY feature, purchase1, purchase2
     FOR MAX @purchase1, MAX @purchase2
```

**Figure 2: Example Business Scenario**

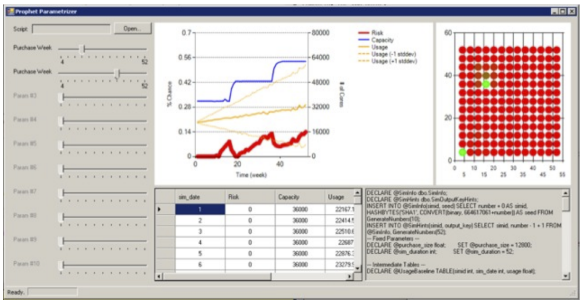

**Figure 3: Fuzzy Prophet's online interface.**

feature release date, representing additional demand resulting from the released feature.

The `Capacity Model` is is expressed as an aggregate of many different individual models, each expressing different classes of hardware failures, as well as expected time from new hardware purchase to deployment. The model accepts a set of hardware purchase dates, constructs (stochastically) a series of events that modify the number of cores available during a given week, and tracks the sum of all changes over the course of the entire year. Note that these table generating functions are stored in the database. If an analyst develops a better model, she can update all all Fuzzy Prophet instances using the model by simply modifying the function definitions.

We demonstrate the use of Fuzzy Prophet in two modes. In online mode we declare `@current` to be the graph's X-Axis. Fuzzy Prophet is asked to graph the chance of insufficient capacity, and the expectations of system capacity and user demand. In offline mode, results are computed for the entire parameter space, and the query returns the latest purchase dates that keep (over the entire year being simulated) the expected chance of overload below 5%.

## 3.2 Fuzzy Prophet in Online Mode

Guests are invited to arbitrarily set the values of purchase date parameters by adjusting the sliders in Fuzzy Prophet's GUI. The GUI is shown in Figure 3. The simulation goal is be to evaluate the business scenario described above with the given parameter values. At first, Fuzzy Prophet takes a few dozen seconds to generate accurate statistics of the output and an accurate rendering of the graph. However, when guests perform a second adjustment, only portions of the graph changed by the adjustment are re-rendered (implying that only a small portion of the output statistics is recomputed).

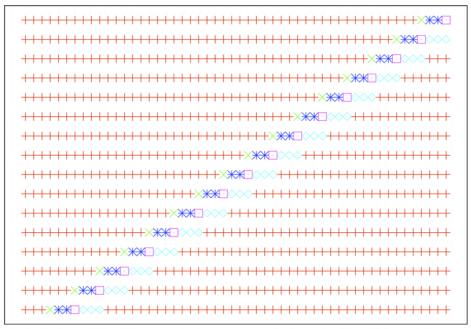

**Figure 4: Visualization of a 2D slice of Fuzzy Prophet's fingerprint mappings for the Capacity model.**

Users are also encouraged to note the effects of changing the feature release date. Fuzzy Prophet's distribution mapping capabilities are able to reduce the set of weeks for which the query must be recomputed, despite the slope of the usage graph changing.

The GUI also shows the small fragment of SQL code required to describe the scenario. In addition, it also shows the parameter space (in form of a 2D grid), with which parameter values have already been explored and which values are proactively being explored anticipating their future usage.

## 3.3 Fuzzy Prophet in Offline Mode

We also demonstrate how Fuzzy Prophet is invoked with the above scenario in a offline mode. The simulation goal is to determine the parameter values that minimizes the total cost of ownership while keeping the risk of overload under a threshold. Guests are invited to vary the simulation characteristics (e.g., starting the simulation with a different initial capacity or a different user growth). Guests observe as the query runs to completion, as a live-updated view shows the simulation's progress through the parameter space, as well as any established mappings, as in Figure 4.

# 4. REFERENCES


[1] L. Antova, C. Koch, and D. Olteanu. MayBMS: Managing incomplete information with probabilistic world-set decompositions. In *ICDE*, 2007.

[2] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *ACM SIGMOD*, 2005.

[3] D. Hamlet. *Encyclopedia of Software Engineering*, chapter Random testing, pages 970–978. Wiley, New York, 1994.

[4] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. MCDB: a monte carlo approach to managing uncertain data. In *ACM SIGMOD*, 2008.

[5] O. Kennedy and C. Koch. PIP: A database system for great and small expectations. In *ICDE*, 2010.

[6] O. Kennedy and S. Nath. Jigsaw: Efficient optimization over uncertain enterprise data. In *ACM SIGMOD*, 2011.

[7] M. Mutsuzaki, M. Theobald, A. de Keijzer, J. Widom, P. Agrawal, O. Benjelloun, A. D. Sarma, R. Murthy, and T. Sugihara. Trio-One: Layering uncertainty and lineage on a conventional dbms (demo). In *CIDR*, pages 269–274, 2007.

[8] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. E. Hambrusch, and R. Shah. Orion 2.0: native support for uncertain data. In *ACM SIGMOD*, 2008.