Proceedings of the 2009 INFORMS Simulation Society Research Workshop
L. H. Lee, M. E. Kuhl, J. W. Fowler, and S. Robinson, eds.

DATABASE MEETS SIMULATION: TOOLS AND TECHNIQUES

Peter J. Haas

IBM Research Division
Almaden Research Center
San Jose, CA 95120-6099, U.S.A.

Christopher Jermaine

Department of Computer Science
Rice University
Houston, TX 77251, U.S.A.

ABSTRACT

We describe some recent technology for Monte Carlo analysis within the setting of a database management system. The original motivation for this work was the desire to quantify the uncertainty in the answers to database queries when the underlying input data is uncertain due to data integration, information extraction from text, and so forth. It has since become apparent that our prototype systems can also be used to process data-intensive queries in which uncertainty arises from the use of stochastic models to extrapolate missing or hypothetical data. Our systems then permit complex analysis and query processing without the need to continually transfer the data between the database and a simulation package. Our use of the map-reduce processing framework allows robust, massively parallel simulation processing on commodity hardware. We outline some key ideas and technical challenges that arise in our novel Monte Carlo environment.

1 INTRODUCTION

This paper describes a recent convergence of database and simulation technologies to support a type of Monte Carlo analysis that can be described as "data-intensive simulation." The original motivation for this work arose in the database community, specifically, researchers working in the field of "probabilistic databases;" see Dalvi and Suciu (2007) for a recent survey. The focus of this initial work was on relational database management systems (RDBMS) in which the underlying data is uncertain. In this setting, the answer to a query is not a single number or set of records, but rather a distribution over possible query answers; the goal is to compute interesting features of this distribution, such as means, quantiles, or record-inclusion probabilities.

One common source of data uncertainty arises in data integration, where data from heterogeneous sources is combined into a single data warehouse; such a scenario is becoming increasingly common as enterprises combine new and legacy data sources, often as a result of mergers and acquisitions. In this setting, records corresponding to a given entity, such as a customer, need to be "resolved" and merged into a single record. Values of specified attributes, e.g., city of residence, might differ among these records, leading to uncertainty. Another source of data uncertainty results from the extraction of structured data from unstructured text; see Michelakis et al. (2009) and references therein. When, e.g., a span of text has been annotated as a "Person," there is typically some uncertainty about whether the span truly refers to a person, and hence whether the resulting extracted record truly belongs in the database.

Over the past few years, probabilistic database systems have been developed to specify uncertainty in the data warehouse and to quantify the impact of this uncertainty on answers to queries over this warehouse data; see Xu et al. (2009) and references therein. Most of these systems have emphasized exact computation of record-inclusion probabilities for the query answer, with the goal of displaying the K most likely records. This exact-computation approach places strong restrictions on the types of uncertainty that can be modeled, the class of queries that can be handled, and on the characteristics of the query answer that can be evaluated. For example, such systems have trouble handling "aggregation" queries that compute sums, averages, and so forth, and yet such queries are central to enterprise decisionmaking.

To deal with the drawbacks of existing probabilistic databases, Jampani et al. (2008) developed a Monte Carlo database system (MCDB). In this system, the user characterizes uncertainty by specifying "value generation" (VG) functions that sample from the distributions of uncertain data values. Instead of computing a query-answer distribution exactly, MCDB uses the VG functions to draw i.i.d. samples from this distribution, and then uses standard techniques to estimate distribution features. This approach permits a highly flexible and extensible uncertainty model, processing of arbitrarily complex queries, and estimation of a broad collection of functionals of the query-answer distribution.

CustName	OptionID	NumShares
"Jane Smith"	23	50
"Jane Smith"	108	100
"Fei Xu"	38	500
...	...	...

Customer

OptionID	InitVal	r	a	StrikeP	OVal
23	\$2.35	0.8	1.01	\$4.00	\$5.23
38	\$5.00	0.5	1.12	\$3.00	\$8.70
...	...	...	...	...	...

Option

Figure 1: A simple relational database.

Perhaps more importantly, it has become apparent that MCDB can also handle the uncertainty that arises when key data needed to answer a query are not present in the database in any form at all, and must be extrapolated using a stochastic model. In the sequel, we give an example of extrapolation uncertainty in the context of a simple financial model.

The MCDB computations can be highly CPU intensive, but offer the potential for massive parallelization. To realize this potential, Xu et al. (2009) have provided a new system called MC^3 (Monte Carlo Computation on a Cluster) that extends the MCDB approach to the map-reduce processing framework. MC^3 can exploit the robustness and scalability of map-reduce, which allows reliable massively parallel processing over commodity hardware. A key issue that arises in this context is management of pseudorandom seeds in a distributed and highly parallel fashion.

In the following sections, we give an overview of some basic technical ideas and challenges underlying the MCDB and MC^3 prototypes. We also indicate directions for future research that are pertinent to the simulation community.

2 A Monte Carlo Database System

2.1 Relational Databases

The MCDB system can be viewed as "wrapping" the classical relational database model with an extensible model of uncertainty. We briefly review relevant relational database concepts here; see, e.g., Garcia-Molina et al. (2009). A relational database is a collection of *relations*, or tables, that are stored on disk; a few such tables are depicted in the financial database of Figure 1. In the Option table, each row corresponds to a European call option, and the columns correspond to attributes of the option (described below); a similar organization holds for the Customer table, which records the option holdings for a set of customers, with one row per distinct (option, customer) pair. The *schema* of a table is a list of the attributes that comprise the table; in standard relational terminology, $T.a$ denotes attribute a in table T. A database *query* describes a desired output relation which is constructed from the base relations that are stored on disk. The Structured Query Language (SQL) is used to specify a relational query. For example, the following SQL query computes the value of Jane Smith's portfolio:

```
SELECT SUM (c.NumShares * o.OVal) as PValue
FROM Customer AS c, Option AS o
WHERE c.OptionID = o.OptionID
    AND c.CustName = 'Jane Smith'
```

The query returns the desired answer as a one-row table having a single attribute, called PValue. This result table is formed by selecting the rows in the Customer table corresponding to Jane Smith, and joining each of these rows to the corresponding row in the Option table (by matching up the OptionID values). Finally, the contribution from each of these joined rows to the total portfolio value is computed and added to the sum. The RDBMS "query optimizer" automatically determines a good "query plan" for actually producing the desired result table.

2.2 MCDB Overview

MCDB, like most probabilistic-database prototypes, is based on "possible worlds" semantics for uncertain data. Each possible world corresponds to a possible concrete realization of all uncertain values in the database, and there is a probability distribution over the set of possible worlds, i.e., over the set of possible database instances. Usually, this possible-worlds distribution is implicitly determined by the probability distributions that are associated with individual (sets of) uncertain attributes; in general, there can be statistical dependencies among attributes both within and between rows.

For example, consider the SQL query given above, but now suppose that the option value (i.e., the Option.OVal attribute) represents the future value of the option at its exercise time. (For simplicity, assume that all options in the table are purchased at time $t = 0$ and exercised at some fixed time $t = T$.) The future value of the option is clearly uncertain, and must be estimated via a model. E.g., we might use the following (very) simple Black-Scholes-type model with varying volatility to model the evolution of $S_i(t)$, the value at time t of the stock that underlies option i: $dS_i(t)/S_i(t) = r_i\,dt + a_i\big(S_i(t)\big)^{1/2}\,dW_i(t)$, where $r_i > 0$, $a_i \in (0,1)$, and $W_1, W_2, \ldots$ are i.i.d. standard Brownian motions. Then the option value V_i at exercise time T is $V_i = \max\big(S_i(T) - k_i, 0\big)$, where k_i is the strike price for the option (recorded in the StrikeP column in Figure 1). This model cannot be solved analytically, but samples from the (approximate) distribution of $S_i(T)$, and hence V_i, can be generated by using a simple Euler-scheme recursion: for a large value of m, fix $\Delta t = T/m$ and set $S_i(t_j) = S_i(t_{j-1}) +$

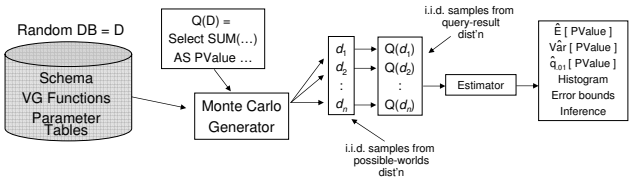

Figure 2: Query evaluation in MCDB.

$r_i S(t_{j-1})\Delta t + a_i\big(S_i(t)\big)^{1/2} S_i(t_{j-1})\sqrt{\Delta t} Z_{ij}$ for $j = 1, 2, \ldots, m$, where $t_j = j\Delta t$ and each Z_{ij} is an independent $N(0,1)$ variate. The parameters for this scheme comprise columns 2-5 of the Option table. Our goal is now to estimate features of the query-answer distribution, such as means or quantiles of the portfolio value.

MCDB does not try to compute characteristics of the query-result distribution exactly, but rather estimates these characteristics using a Monte Carlo approach. Figure 2 shows, conceptually, how MCDB evaluates a query. Each random table in an uncertain database is represented on disk by its schema, together with a set of VG functions that are used to generate realizations of uncertain attribute values, and an optional set of *parameter tables* (ordinary relational tables) that are used to parameterize the VG function. Conceptually, when a query is issued, the MCDB system invokes the VG functions to generate a set of independent and identically distributed (i.i.d) samples from the possible-worlds distribution. The query is then evaluated over each of the sampled possible worlds, thereby generating a set of i.i.d. samples from the query-result distribution. These latter samples can be used to estimate characteristics of interest for this distribution.

Consider, for example, the foregoing portfolio-value scenario with 1000 Monte Carlo iterations (i.e., 1000 sampled possible worlds). In MCDB, two parameter tables, Customer and OptionParam, would reside on disk, where Customer is as in Figure 1 and OptionParam is almost identical to the Option table in Figure 1, except without the OVal column. We could then define a schema for a random table called R-Option having attributes OptionID and OVal. The MCDB schema for this random table essentially gives a recipe for generating a sample realization of the table, specifying which VG function to invoke and how to glue the outputs of the VG function invocations together:

```
CREATE TABLE R-Option(OptionID, OVal) AS
 FOR EACH d IN OptionParam
  WITH OResult AS GenOptVal(
   VALUES(d.InitVal, d.r, d.a, d.StrikeP))
 SELECT d.CID, r.VALUE1 FROM OResult AS r
```

We employ the user-defined VG function GenOptVal, which executes the Euler scheme to generate a sample of a random option value at time T. The GenOptVal function takes as input a 1×4 table that specifies the option parameters needed for the Euler simulation, and returns the result as a one-row table (here named OResult) having a single attribute called VALUE1. The `FOR EACH` clause instructs MCDB to loop over the list of options stored in the OptionParam table. For each option, a realization of the OVal attribute is generated via a call to GenOptVal, which is first parameterized by executing a (trivial) SQL subquery that extracts the appropriate parameter values from the OptionParam table and uses the SQL `VALUES` function to package these values as a 1×4 table. The final `SELECT` clause specifies how to assemble the row of the R-Option table corresponding to the current option from the outer loop.

In general, a VG function takes as input one or more parameter tables, which can be specified via arbitrary SQL queries, and returns a table as output; users can define their own VG functions using a C++ interface similar to that of user-defined functions in a standard RDBMS.

Conceptually, MCDB will follow the above recipe to generate 1000 realizations of the R-Option table and apply the SUM query given previously (but with Option replaced by R-Option) to each realization to obtain 1000 realizations of Jane Smith's portfolio value. Point estimates and confidence intervals for, e.g., the mean value and standard deviation of the future portfolio value can be computed using standard methods. Alternatively, we may simply want to create a histogram of the 1000 numbers in order to get an approximate picture of the portfolio-value density function.

2.3 MCDB Query Processing

MCDB does not actually instantiate a database multiple times; the costs for such a naive approach would be exorbitant. Instead, MCDB executes a single query plan over a set of *tuple bundles*; a tuple bundle represents a row of a random table as it appears in each of the Monte Carlo replications. In our running example, the tuple bundle t in table R-Option corresponding to option 105 might have the form $t = (105, (123.50, 274.00, ..., 180.76), \text{Seed})$. Here the first component is simply the value of the deterministic OptionID attribute, and the second component is a vector that contains the 1000 values of the OVal attribute for option 105 that were generated during the 1000 Monte Carlo repetitions. The final component, Seed, is the pseudorandom number seed used by the GenOptVal VG function to generate the option values. More generally, if customer 105 appeared in some, but not all, of the 1000 possible worlds, the tuple bundle would be augmented by a special random vector called isPresent. For $1 \le i \le 1000$, the ith component value of this vector is *true* if and only if customer 105 appears in the ith sampled possible world.

Because pseudorandom number generation is a deterministic process when started from a fixed seed, the foregoing tuple bundle can often be stored in compressed

Query	1 iter	10 iters	100 iters	1000 iters
Q1	25 min	25 min	25 min	28 min
Q2	36 min	35 min	36 min	36 min
Q3	42 min	45 min	60 min	214 min

Table 1: MCDB running times.

form as $t = (105, \text{Seed}, \text{isPresent})$. The OVal values can be regenerated in a consistent manner whenever they are needed. MCDB attempts to maintain tuple bundles in compressed form whenever possible, to reduce the amount of data that is moved through the system. Moreover, MCDB attempts to apply predicates on deterministic attributes (e.g., `OptionID < 105`) to compressed tuple bundles as early in the query plan as possible; if such predicates filter out most tuple bundles, then most of the Monte Carlo computations can be avoided. MCDB exploits these ideas to achieve acceptable processing overheads.

For example, Table 1, taken from Jampani et al. (2008), shows running times for three relatively complex queries over a 20GB database, involving joins of large tables, expensive VG-function evaluation, grouping, and aggregation. As can be seen, for queries Q1 and Q2, the time to run 1000 Monte Carlo replications is almost the same as the time to run a single replication; most of the processing time is spent in standard RDBMS operations such as disk reads and writes. For Q3, the Monte Carlo replications add overhead to the processing time, but the overhead is far less than a factor of 1000, as would be the case with the naive approach.

MCDB extends the classical database operations of select, project, join, duplicate removal, and aggregation to handle tuple bundles; these extensions typically involve manipulation of the isPresent vector. New MCDB operators include Instantiate and Seed. The Instantiate operator invokes the VG functions to transform a tuple bundle from compressed to expanded form. The Seed operator attaches one or more seeds to a tuple bundle, one per VG function. Each seed is unique to the (tuple bundle, VG function) pair, and is used by the VG function to generate a stream of pseudorandom numbers during instantiation.

3 PROCESSING MCDB QUERIES ON A CLUSTER

A mentioned above, the MC^3 system (Xu et al. 2009) is a re-implementation of MCDB in Hadoop, an open-source implementation of Google's map-reduce processing framework (Hadoop 2009). Hadoop vastly simplifies development of parallel programs, and the system provides a high level of fault tolerance and the capability to allocate and reallocate resources (CPU, memory, storage) as needed.

MC^3 uses Javascript Object Notation (JSON) as the internal data model (JSON 2009). JSON is perhaps the most commonly used data format for map-reduce processing, and is quite flexible. The two basic constructs are ordered arrays and records, the latter comprising a set of (key, value) pairs. Because these data structures can be nested in an arbitrary manner, it is straightforward to represent MCDB tuple bundles in JSON. One advantage of a JSON-based system is that it can potentially handle data that is not in strict relational form, due to nesting or to varying numbers of attributes per record.

3.1 Map-Reduce Processing

A map-reduce job takes as input a collection of key-value records and produces a collection of output values. To specify the desired data processing, the user defines a *map* function and a *reduce* function, whose operation is described below. In the first step of map-reduce processing, the input records are partitioned among one or more mapper tasks. Each mapper applies *map* to the input records in its partition, one record at a time. The *map* function takes as input a key-value record (k, v) and produces a collection of intermediate key-value records, $[(k'_1, v'_1), \ldots, (k'_n, v'_n)]$. For each distinct intermediate key k, all intermediate records having this key are collected to form a group $(k, [v''_1, \ldots, v''_m])$. These groups are partitioned among a set of reducer tasks. Each reducer applies *reduce* to the groups in its partition, one group at a time. The *reduce* function takes a group as input and returns a final value for the group, often using an aggregation operation on the values in the group. The set of values returned by the calls to *reduce* is written out and comprises the output of the map-reduce job.

Both mapper and reducer tasks operate in parallel. Xu et al. (2009) discuss a couple of parallelization schemes. The simplest scheme is *inter-tuple* parallelism, which is obtained by evaluating multiple tuple bundles in parallel; all of the possible worlds for a tuple bundle are generated by a single CPU core. Preliminary experiments indicate good scale-out properties using this scheme. E.g., for the queries in Table 1, the processing time remains almost constant when both the number of processors and the database size are scaled up by an order of magnitude.

Queries in MC^3 are specified using JAQL, a high-level query language designed for JSON data (JAQL 2009). The key appeal of JAQL is its ability to automatically generate a query plan of map-reduce jobs from a high level query specification. MC^3 makes extensive use of JAQL's user-defined functions to specify VG functions, distributed seeding, and tuple-bundle operations.

3.2 Distributed Seeding

Seeding tuple bundles is challenging because it must be done in a highly parallel and distributed fashion, over a possibly enormous number of tuple bundles, and without requiring storage of too much seeding information in each tuple bundle. The seeding scheme must also guarantee that distinct

tuple bundles use disjoint sequences of seeds during the instantiation process, in order to avoid unintended statistical dependencies; i.e., the scheme must avoid "collisions." MC^3 supports two distributed seeding methods: SeedSkip, which is based on skip-ahead capabilities for (typically huge-period) pseudorandom number generators (PRNGs), and SeedMult, which uses multiple generators.

The SeedSkip method uses a single PRNG for both seeding and instantiation — chosen as WELL512 (Panneton et al. 2006) in the current implementation — and requires that we can upper bound the number of bundles seeded per mapper and the number of seeds used up during any single invocation of any VG function. SeedSkip trivially "seeds" a tuple bundle with its mapper number and intra-mapper tuple-bundle identifier. To naively instantiate a bundle, we would use this information to generate an appropriate number N of fixed-length skips along the PRNG's cycle to reach the starting point for the instantiation. The number of skips can be reduced to $O(\log_2 N)$ by precomputing $O(\log_2 N)$ sets of "skip information" corresponding to a set of exponentially increasing skip lengths.

The SeedSkip method cannot be used if the VG functions are so complicated that it is impossible to obtain an upper bound on the number of seeds consumed during a VG function call. In this case, the SeedMult seeding method can be used instead. Although SeedMult is not guaranteed to avoid collisions, the probability of collisions can be made vanishingly small; see Xu et al. (2009) for details.

4 FUTURE DIRECTIONS

The MCDB and MC^3 prototypes represent a convergence of simulation and database technology, potentially opening up new applications for simulation modeling and analysis. There are many possible directions for future research.

A key challenge is to support both the standard and new functionality that is typically provided by a simulation package. For example, in the current prototypes, the user must specify the desired number of Monte Carlo iterations, which can be hard to do without guidance. The goal here is to have the user specify precision and/or time requirements, and have the system automatically determine the number of iterations, perhaps dynamically. Closely related to this issue is the question of how to define an appropriate syntax for specifying the functionals of the query-output distribution required by the user, along with the speed and precision requirements. Also of interest is the question of how to incorporate well known variance reduction schemes, as well as methods for stochastic optimization.

For purposes of risk assessment, we often want to estimate quantiles of the distribution of a query result. This task can be challenging for extreme quantiles that correspond to rare events. An interesting question is how to apply techniques such as importance sampling (Henderson and Nelson 2006, Ch. 11) in the current setting; the problem structure appears to be quite different than for, e.g., classical buffer-overflow problems. Importance sampling can also potentially be used to "push down" selections into the VG function, i.e., to only generate sample rows that satisfy selection predicates; see Xie et al. (2008).

For the MC^3 system, one important problem is how to specify and efficiently process uncertain JSON data; MC^3 is currently limited to nested JSON data for which only leaf values are uncertain. Finally, an intriguing feature of Hadoop is the potential for implementing dynamic simulation techniques, since Hadoop resources can be dynamically re-allocated. E.g., as in Schruben (2008), it may be possible to simultaneously simulate competing business policies, redirecting resources away from a policy as soon as it becomes apparent that the policy is inferior.

REFERENCES

Dalvi, N. N., and D. Suciu. 2007. Management of probabilistic data: foundations and challenges. In *Proc. PODS*, 1–12.

Garcia-Molina, H., J. D. Ullman, and J. Widom. 2009. *Database Systems: The Complete Book*. Second ed. Prentice Hall.

Hadoop 2009. Available via <http://hadoop.apache.org/core> [accessed April 29, 2009].

Henderson, S. G., and B. L. Nelson. (Eds.) 2006. *Simulation*. North-Holland.

Jampani, R., F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. 2008. MCDB: a Monte Carlo approach to managing uncertain data. In *Proc. SIGMOD*, 687–700.

JAQL 2009. Available via <http://code.google.com/p/jaql> [accessed April 29, 2009].

JSON 2009. Available via <http://www.json.org> [accessed April 29, 2009].

Michelakis, E., P. J. Haas, R. Krishnamurthy, and S. Vaithyanathan. 2009. Uncertainty management in rule-based information extraction systems. In *Proc. SIGMOD*. To appear.

Panneton, F., P. L'Ecuyer, and M. Matsumoto. 2006. Improved long-period generators based on linear recurrences modulo 2. *ACM Trans. Math. Software* 32 (1): 1–16.

Schruben, L. 2008. Analytical simulation modeling. In *Proc. Winter Simulat. Conf.*, 113–121.

Xie, J., J. Yang, Y. Chen, H. Wang, and P. Yu. 2008. A sampling-based approach to information recovery. In *Proc. ICDE*, 476–485.

Xu, F., V. Ergcegovac, P. J. Haas, and E. Shekita. 2009. *E = MC^3*: Managing uncertain enterprise data in a cluster-computing environment. In *Proc. ACM SIGMOD*. To appear.

AUTHOR BIOGRAPHIES

PETER J. HAAS is a research staff member at the IBM Almaden Research Center and a consulting associate professor in the Department of Management Science and Engineering at Stanford University. He is vice president of the INFORMS Simulation Society, an area editor (Stochastic Models) for *ACM TOMACS*, and an associate editor (Simulation Area) for *Operations Research*. His research interests include discrete-event simulation, as well as the application of stochastic methods to problems in information management. His web page is <http://www.almaden.ibm.com/cs/people/peterh>.

CHRISTOPHER JERMAINE is an associate professor of computer science at Rice University. He is an associate editor for the *Very Large Databases Journal*, and is the recipient of a 2008 Alfred P. Sloan Foundation Research Fellowship, a National Science Foundation CAREER award, and a 2007 ACM SIGMOD Best Paper Award. His research interests focus on database systems, and his work has a strong statistical flavor. His email address is <cmj4@cs.rice.edu>.