Data Access Pattern Analysis and Prediction for Object-Oriented Applications

STOYAN GARBATOV[1]
JOÃO CACHOPO[2]

Software Engineering Group
Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento, INESC-ID
Lisbon, Portugal
[1]stoyangarbatov@gmail.com
[2]joao.cachopo@ist.utl.pt

Abstract. This work presents an innovative system for analysing and predicting the runtime behaviour of object-oriented applications, with respect to the data access patterns performed over their domain objects. The analysis and predictions are performed using three alternative stochastic model implementations. The models are based on Bayesian Inference, Importance Analysis, and Markov Chains. The system deals with all the necessary modifications of the target applications under analysis in a completely automatic fashion, without it being necessary for any developer intervention. The results are validated by the execution of the TPC-W and oo7 benchmarks. The oo7 benchmark has been modelled as a stochastic process through Monte Carlo simulations. We show that the results obtained with our system are precise, regarding the observed behaviour, and that the overheads introduced by the data acquisition are low, ranging from 5% to 9%. The system is sufficiently flexible to be applied to a broad spectrum of object-oriented applications.

Keywords: Data access pattern, Persistent data, Stochastic modelling, Bayesian Inference, Importance Analysis, Markov Chains, Monte Carlo.

(Received August 23rd, 2011 / Accepted November 24th, 2011)

1 Introduction

Modern three-tier application architectures place business logic on an application server and the persistent objects on one or more back-end relational database management systems (RDBMS). This architecture, however, poses problems to applications with object models that are inherently navigational: models where objects have references or relationships to other objects, which applications follow one at a time. The navigational characteristics of applications may cause a large number of data-access round trips, which could result in unacceptable overhead due to the latency from both physical disk access and network processing. This overhead, which is due to inter-process communication costs, is becoming increasingly important because of the recent gains in the number of requests that can be processed per second due to improvements in processor speed, memory size, disk throughput, and query processing algorithms.

The current state-of-the-practice to minimize this problem is to make programmers responsible for carefully tuning an application with the aim of reducing the number of database round-trips required to fulfil an application´s request for stored objects. Yet, this tuning is hard to do and often leads to an excessive amount of time spent in the development of an application. An additional drawback of this approach is that such manual tuning must be repeated each time the database schema or application logic is modified. Such modifications are quite common because of business or technical

progress.

Thus, we argue that the current approach of manual tuning is hindering the development of complex applications and is untenable in the long run. Instead, we intend to automate that tuning, by developing self-adaptable mechanisms of data prefetching and caching that take into account the probabilistic behaviour of an application. With these mechanisms in place, we expect to minimize the user-perceived elapsed time of tasks, given that the elapsed time is strongly related to the number of DBMS round-trips and these mechanisms seek to minimize them. In the work reported in this paper, we do not address the implementation of these self-adaptable prefetching and caching mechanisms, which we intend to do in a future work, but we address what we believe to be an essential pre-requisite for such work: the ability to identify the data access patterns performed by an object-oriented application and to predict with precision what the future behaviour of that application will be, with regards to the domain object manipulations that it performs.

In particular, we describe an automatic system that is able to monitor the data access patterns of an object-oriented application during its execution, analyse those patterns using several models, and predicting what types of domain objects are most likely to be accessed during the forthcoming execution of the application.

The system is expected to satisfy a set of requirements, among which are the following:

- Efficiency - the system should be able to acquire and analyze the necessary statistical data in an efficient manner and with the minimal possible amount of overheads for the target application;
- Transparency - from the point of view of the application programmer, the system should be able to achieve its goals with minimal or no intervention by the application programmers;
- Precision - the predictions generated by the system should be as close as possible to the observed behaviour by the applications, in the future.

The article has the following structure. Section 2 discusses the related work. Section 3 describes the main structure of the system that we propose. Then, in Section 4, we present the stochastic models that were employed in this work, where their theoretical background is considered first, followed by a description of their actual implementation. The results and evaluation of the system are discussed in Section 5. In Section 6, we make a brief analysis and comparison of the three stochastic models, and in Section 7, we analyse the overheads introduced into the target application by our system. Finally, in Section 8, we present some concluding remarks about our work and the results obtained.

2 Related Work

Apart from the work reported in [11, 12], to the best of our knowledge, there is no previous published work that uses stochastic methods to predict the behaviour of object-oriented applications, in what concerns the domain data manipulations they perform in runtime. There is, however, some research work on using stochastic models for traffic analysis in telecommunications that has some relation to the approach described in this paper. We start by briefly reviewing that work, and then we describe some of the previous work for optimizing application performance that we envision may use the predictions generated by the system that we present here.

A study for telecommunication traffic with regards to the use of stochastic models has been presented in [24, 25, 23]. In that field of research, many quantities of interest have been shown to have a long tailed distribution. A long-tailed or heavy-tailed probability distribution is one that assigns relatively high probabilities to regions far from the mean or median. It should be noted that the long-tailed assumption dominant in the works does not apply at all for the current work. However, these works are relevant from the point of view of their use of stochastic model analysis. In particular, Li [25] studied the stochastically bounded modelling of traffic by taking into account both stochastic and deterministic modelling parameters resulting thus in a new approach for analysing traffic behaviour. Also, Li [24] considered the traffic modelling as a generalized Cauchy process, leading to a flexible approach capable of describing multi-fractal traffic phenomena, as well as accurate traffic modelling.

Concerning performance optimizations that can be attempted when there is available information about what are the most likely domain data sets to be accessed, there are a variety of approaches. Two categories of such methodologies, where significant research work has been performed, are caching and prefetching. Caching refers to storing locally in memory recently accessed objects, thereby avoiding unnecessary requests to the database [8].[28] and [17]. Caching is useful in many applications, but it does not help if future requests rarely match previous ones. Prefetching, which is about fetching data based on a prediction of an applicationś future requests, can help in such situations [33], [16] and [35]. Both caching and prefetching may result in significant pay-offs in data-

access performance [20] and [1]. For instance, some studies of prefetching reported improvements of several times by prefetching multiple tuples at a time instead of just the one that is requested [4].

Prefetching techniques may be classified into three main categories [19]:

- Deterministic prediction techniques employ a fixed strategy and are used by commercial data servers and by object managers or containers in application servers. An example strategy is to load one block ahead in sequential prefetching [15], [27] and [13].

- Object structure-based prefetching techniques are mostly used in OODBMSs and predict future data accesses via pointers from objects to other objects [14], [16] and [4].

- Statistical prediction techniques generate probabilistic information about future accesses by analysing past accesses [31], [22], [10] and [29].

Palmer [31] proposed Fido, which is a system based on recognizing patterns of object references to predict future references. The authors used an associative memory to predict future requests based on access patterns that approximately match a pattern in the training trace.

Lei [22] monitored the files used by programs in order to build an access tree that encapsulates the program file reference behaviour. When the current access tree matches one of the previous access trees (within some threshold), the stored tree is used to prefetch all of the files that are anticipated to be needed for the program.

Madhyastha [29] employed neural networks and a hidden Markov model to predict future client behaviour, in terms of input/output patterns. The authors conclude that the Markov model offers a more precise control over caching and prefetching policies than neural network access pattern classification. Drapeau [10] proposed using data mining techniques to find association rules for prefetching in WWW search engines, where an off-line process identifies frequent item sets to build a prediction model using earlier requests to predict future requests.

3 System Description

The system is composed of three modules: a code injection module, a data acquisition module and a data analysis module. The code injection module is responsible for injecting code into the target applications. The injected code is responsible for invoking the functionality present in the other two system modules and its injection is performed in a completely automatic fashion by the system to avoid the need for the application programmers themselves to perform any modifications whatsoever to their applications. The second module - the data acquisition module - is responsible for collecting data about the actual behaviour of the target application, with regards to the data accesses that the application performs. Finally, the data analysis module is responsible for applying the stochastic models over the acquired data and, subsequently, for generating the prediction about the most likely behaviour to be observed by the target application, in terms of the domain objects that it will access. An overview of the runtime behaviour of the system can be seen in Figure 1 (arrows indicate data flows and rectangles correspond to modules).

3.1 Code Injection

The data-access patterns of an application describe which types of domain objects and which of their fields are accessed during its execution. Consequently, to capture these patterns, it is necessary to identify all the points where data is accessed within the application and instrument them. This allows the subsequent recording of those accesses during the execution of the application.

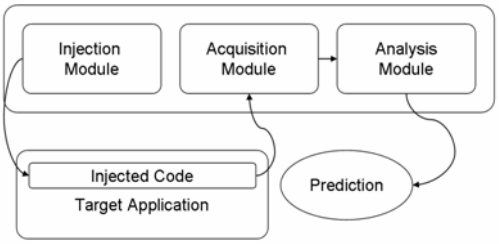

Figure 1: System runtime overview

The system performs this instrumentation at compile-time by means of (Java) byte-code rewriting. Thus, the system performs all the necessary modifications to the target application in an automatic way, without the intervention of the programmer. The modifications are achieved through two instrumentation phases, both of which make use of the Javassist library to inject the code.

A relevant research work in this field has been performed by Whaley [34], where a run-time system for performance analysis of applications running on a Java

virtual machine is presented. The system provides detailed and continuous system performance information to a dynamic compiler, with the goal of allowing the compiler to make better decisions when performing code optimizations.

The first instrumentation phase injects the code necessary for the identification of the contexts within which all data accesses take place. The context is a key concept in the system. It defines the scope within which all data manipulations take place. For the results presented in this paper, the context corresponds to the method in which body the data accesses take place, but the system supports different context definitions. For instance, the context can correspond to the execution of a given service or even a sequence of the last n method invocations that precede the current point of the target application execution.

There are several reasons that led us to the use of the simpler context definition instead of more elaborate ones. The first of these relates to the fact that whilst a more complex context definition could, potentially, bring more information to the inference models, it would increase the performance overhead due to the collection of application behaviour information. As one of the system requirements is efficiency at the level of gathering statistical information, the most performance friendly context definition was employed.

Another consideration is that the use of a context defined as the "n preceding method invocations present in the execution stack" would make the space of the "possible" contexts grow exponentially, as a function of the stack depth parameter n. This would lead not only to significantly higher upper memory bounds for the storage of the statistical information (discussed in more detail in the end of the current section), but it would also increase the time interval necessary for the collection of enough statistical data for the stochastic inference models to be able to generate their predictions. This last issue shall be discussed further in Section 4.

Listing 1: Method source code before instrumentation

```
void addPart(Part p) {
 if (this.parts == null) {
  this.parts = new HashSet<Part>();
 }
 p.setCompositePart(this);
 this.parts.add(p)
}
```

To identify the contexts at runtime, the first instrumentation phase modifies each method of the application. It injects code that updates the context information associated with the method, upon entering it, and code that cleans the same context information when returning from it.

An example of the source code of a given method, before it is instrumented by this phase, can be seen in Listing 1, whereas the resulting (decompiled) code, obtained after the first instrumentation, can be observed in Listing 2.

It should be noted that the *renamedAddPart* method corresponds to the original method that has been renamed. This is shown in Listing 3.

Listing 2: Method source code after first phase

```
void addPart(Part p) {
 ContextManager.addContextInfo(
  new Info("oo7.CompositePart",
           "addPart"));
 renamedAddPart(p);
 ContextManager.removeContextInfo();
}
```

Listing 3: Renamed method body

```
public void renamedAddPart(Part p) {
 if (this.parts == null) {
  this.parts = new HashSet<Part>();
 }
 p.setCompositePart(this);
 this.parts.add(p)
}
```

The second instrumentation phase replaces every field access that exists within the application with the invocation of a previously injected static method. There is a distinct method for each of the fields for every class of the application. These methods determine the surrounding context in which the field is accessed and update the associated statistical information. This is done according to the type of access performed: a distinct method is invoked whether the field is read or written.

Listing 4: Sample results of the second phase

```
void renamedAddPart(Part p) {
 if (staticReader_parts(this) == null)
   staticWriter_parts(this,
                 new HashSet<Part>());
 p.setCompositePart(this);
 atomicReader_parts(this).add(p);
}
```

With this in consideration, the code presented in Listing 3 suffers modifications from the second instrumentation phase. The results of this are shown in Listing 4.

As can be seen from Listing 4, the read and write accesses to the field parts have been replaced by calls to the methods *staticReader_parts* and *staticWriter_parts*. The bodies of these two methods can be seen in Listing 5 and Listing 6, respectively.

Once these two phases are complete, the application can be put into operation, and it will automatically collect the statistics about each data access, without any further modifications.

Listing 5: Field reader method body

```
static Set<Part> staticReader_parts(
    CompositePart part) {
 ContextManager.updateReadStatistics(
    "oo7.CompositePart", "parts", 1);
 return part.parts;
}
```

Listing 6: Field writer method body

```
static void staticWriter_parts(
    CompositePart part, Set<Part> set){
 ContextManager.updateWriteStatistics(
    "oo7.CompositePart", "parts", 1);
 part.parts = set;
}
```

An important aspect of the solution presented here is the data structure used to store the statistical data. During the instrumentation phases, the system performs an analysis of the structure of the target application via reflection. As a result of this analysis, it generates a set of *PClass* instances to model the structure of each of the target application classes. Each of these *PClass* instances contains a set of *PField* instances, which are used to represent each of the fields that exist in the application class. The information kept in a *PField* covers not only the name and type of a field, but also the number of times that it has been accessed in a context. The subsequent probabilistic analysis uses this information to make its calculations.

The relationship between *PClass* instances and an application class is many-to-one. This can be explained with the fact that the *PClass* instances store information regarding the way that application classes are accessed in the contexts during execution. Consequently, if the same class is used in different contexts, distinct *PClass* instances will keep the information about how that class was used in each.

Considering this, it is possible to estimate the upper bounds on the memory requirements for maintaining these structures. The memory will be proportional to the number of domain classes, times the maximum number of fields present in any of the domain classes, times the maximum number of contexts, which, for the current work, corresponds to the number of methods that exist in the target application. More importantly, it should be noted that the memory requirements, after reaching the referred upper bound, will no longer grow, independently of how long the application operates. In general, it has been observed that the consumption of memory is negligible when compared to the memory needed by the application.

Moreover, given that the probabilistic analysis iterates through the *PField* and *PClass* instances, the time spent in the probabilistic analysis will also take time that is proportional to the previously stated bound.

4 Models

This section presents the three stochastic models employed in the current work with the aim of generating predictions with regards to the data access patterns performed by target applications over their domain objects. For each of the models, the most relevant theoretical aspects are considered, apart from their actual system implementations.

It should be noted that the main contribution of this work resides in the way the models were employed for the goal at hand, namely predicting with precision the behaviour of object-oriented applications regarding domain data manipulations. The models themselves have not been extended or otherwise modified from their "standard" definitions in the literature.

4.1 Bayesian Inference

4.1.1 Theoretical Base

Bayesian analysis techniques are used for parameter estimation. They give an estimate of the statistical uncertainty of the estimated parameters (corresponding, in our work, to the likelihood of reading/writing a given field of an application class) and can update them when new information becomes available.

If observations of one (or more) of the stochastic variables $\mathbf{X}$ are available, the probability density function can be updated. Consider a stochastic variable X with a probability density function $f_X(x)$. If $\mathbf{q}$ denotes a vector of parameters defining the distribution for X, the density function of the stochastic variable X can be written as $f_X\left(x, \mathbf{q}\right)$.

It is assumed that n observations (realizations) of the stochastic variable X are available making up a sample $\widehat{\mathbf{x}} = \left(\widehat{x}_1, \widehat{x}_2, ...\widehat{x}_n\right)$. The realizations are assumed to be independent. The updated density function $f''_{\mathbf{Q}}\left(\mathbf{q}\left|\widehat{x}\right.\right)$ of the uncertain parameters $\mathbf{Q}$, given the realizations, is denoted the posterior density function, see textbook on Bayesian statistics, for example in [5] and [26], and is given by:

$$f''_{\mathbf{Q}}\left(\mathbf{q}\left|\widehat{x}\right.\right) = \frac{f_X\left(\widehat{x}\,|\mathbf{q}\right) f_{\mathbf{Q}}\left(\mathbf{q}\right)}{\int f_X\left(\widehat{x}\,|\mathbf{q}\right) f_{\mathbf{Q}}\left(\mathbf{q}\right) d\mathbf{q}} \quad (1)$$

where $f_X\left(\widehat{x}\,|\mathbf{q}\right) = \prod_{i=1}^{n} f_X\left(\widehat{x}_i\,|\mathbf{q}\right)$ is the probability density at the given observations assuming that the distribution parameters are $\mathbf{q}$. The integration in (1) is over all possible values of $\mathbf{q}$.

4.1.2 Model Implementation

The implementation of the Bayesian Inference Model is presented next. Two sets of statistical data are used to generate the predictions. The first set is called *prior* set and it contains data about access patterns observed in the past, up to a given point in time. This time reference corresponds to the moment when the model prediction was updated last. The second set is called *current* set, and it contains data from the point in time when the *prior* set ends, to the moment when the new updated prediction is to be generated.

It should be noted that the model prediction updating is performed at regular time intervals. The exact duration of these intervals is application dependent and should take into consideration several factors. The time intervals should be long enough to allow the accumulation of a representative volume of behavioural data. In other words, sufficient time should be given to the target application to perform an adequate amount of operations to correspond to a sample of typical system operation. Depending on the target application, the duration of the time interval may vary from several minutes to several hours, based on the amount of work being performed by the target system.

If the time periods are too short, the model prediction updating in the end of these (intervals) would lead to two potential problems. The first of these problems is that if the current data set collected during this time period is not representative of the typical application behaviour, when the current set is used to update the model prediction, it may make the model conclude something that is not true about the target system behaviour. The second problem relates to the fact that very frequent recalculations of the model predictions may add up to a performance overhead that could be otherwise avoided.

When the moment for updating the model prediction comes, the probability density functions of the prior and current data sets are determined. These functions describe the behaviour of the target application, in terms of data accesses performed for the periods of time to which the sets belong. Using the current probability density function, the prior function is updated, obtaining thus the so-called posterior probability density function. The posterior function corresponds to the prediction generated by the model, and describes what is the most likely future behaviour of the target application, in terms of the data it manipulates. Based on the posterior function, the actual access probabilities for all of the application domain class fields are calculated.

4.2 Importance Analysis Techniques

4.2.1 Theoretical Base

An Importance Analysis is a procedure for analysing the potential failure modes within a system by classifying them based on the severity or the effect of failures on the system. It is widely used in many industries during various phases of the system life cycle, [9], [18] and [21]. A failure mode can be defined as the manner by which a failure is observed and it generally describes how the failure occurs. Tools used in the design stage for identifying failures and determining their consequences are Risk Priority Numbers (RPN), Occurrence/Severity Matrix (OSM), Risk Ranking Tables (RRT) and Criticality Analysis (CA). For the current work, these methods are adapted to indicate which groups of fields are more critical or important for the operation of the considered target application.

The Risk Priority Number (RPN) system is a relative rating system that assigns a numerical value to the issue in each of three different categories - Severity (S); Occurrence (O) and Detection (D). The three ratings are multiplied together to determine the overall RPN for the issue. The criteria used in each rating scale are determined based on the particular circumstances for the item that is being analyzed. Because all issues are rated according to the same set of rating scales, this number can be used to compare and rank issues within the analysis. However, because the ratings are assigned with regards to a particular analysis, it is generally not appropriate to compare RPN numbers among different analyses.

Failure mode, effect analysis and criticality analysis techniques are used throughout industry for a variety of applications and consist in a flexible analysis method that can be performed at various stages in the system life cycle. These analyses can be employed to support design, development, manufacturing, service and other activities to improve reliability and increase efficiency.

4.2.2 Model Implementation

The implementation of the Importance Analysis model shall be considered next. Based on the gathered access pattern statistical data, the *local* and *global* access probabilities for all domain class fields are calculated. The *local* access probability corresponds to the likelihood of accessing a certain piece of information within the scope of a given *context*, which has been identified to exist during the execution of the target system. The *global* access probability is defined by the likelihood of accessing a given datum at the level of the whole application. Once the *local* and *global* access probabilities are calculated, they are submitted to a normalization process that generates an access probability classification, within which all access probabilities are placed. If a given access probability has the value of *p* , then its associated access probability class *i* would be calculated according to the following formulae:

$$i = floor\left[(p + (p_{max} - p_{min}))\,/\,(p_{max} - p_{min})\right] \quad (2)$$

where $p \in [0, 1]$ and p_{min} and p_{max} correspond to the minimum and maximum access probabilities observed for that classification type (local or global). The local and global access probability class indices are two of the three input arguments used to generate the final result of the Importance Analysis model.

The third and last input is called the *impact factor*. The *impact factor* corresponds to an expert judgement coefficient whose value should be provided by someone who has solid knowledge of how the application operates (e.g. developers). It is a subjective indication of how important a given piece of domain data is for the execution of the target application, from an "expert" point of view. If no *impact factor* is provided, for a given application class field, the implementation of the model provides a default "neutral" value to be used instead.

The RPN value (for a given domain class field) can be calculated by multiplying the local access probability class, the global access probability class and the impact factor. The RPN values give an indication of how important are their respective fields to the execution of the target application.

4.3 Markov Chains

This section deals with the Markov Chain model [30] and its implementation in the current work. The model analysis procedure is composed of several phases. During the first phase, a transition matrix is built. This matrix contains the probabilities of navigating from one system state to another (automata theory). As one of the main goals of this work is to allow the identification and prediction of access patterns performed by object-oriented applications, the states correspond to the manipulation of a given domain class field. In the transition matrix $T = [t_{ij}]$, the cell t_{ij} contains the probability of manipulating field i immediately after having manipulated field j.

To calculate these probabilities, all *contexts* (that have been identified so far) keep a hash table containing the statistical data concerning the sequences of observed field accesses. The keys of the hash table correspond to *PFields*, while the associated values are sets of *PFields*. The *PField* sets contain access information for the situations when they have been accessed immediately after the hash table key (*PField*) to which they are associated.

All the required input for the analysis is collected during a training period. The training is to be performed only once, as long as it is representative of the application service life. During this period, every time there is a field access, the surrounding *context* is determined. Subsequently, the last field that has been accessed in that *context* is identified and used to index the *context´s* hash table containing statistical data about what access sequences have been seen. After this, the data is updated to reflect the current field access and, finally, the last accessed field is updated to the current one. This allows the gathering of data about what fields are accessed after a given one and the number of times this has been observed.

When the accumulated statistical data is representative, the transition matrix can be built. First, a square matrix T of size $n \times n$ is created, where n is the number of fields in all of the application domain classes. This is used as a base for the calculations, which result in the transition matrix, and, subsequently, in the stationary vector.

For each column $j \in [1, n]$ of the matrix T, each of its $i \in [1, n]$ cells contains the number of times that the field i has been accessed immediately after j. A normalization phase is performed next. For a column j, the value of each of its cells is divided by $\sum t_{ij}, i \in [1, n]$, obtaining the normalized transition matrix, $\bar{T}$.

This matrix does not present all the necessary properties in order to be suitable for the *Power* method [2]. This method consists in calculating the powers of the transition matrix to obtain the stationary distribution.

The *Random Surfer Model* [6] is employed to make the matrix suitable. The newly formed matrix is iterated through, identifying the columns whose elements are all zero. For any such column j, its elements are set to

$t_{ij} = 1/n$.

Following this, a perturbation matrix E with size $n \times n$ is constructed. All of its cells have the value of $1/n$. Once this is done, the $\bar{\bar{T}}$ matrix is defined as $\bar{\bar{T}} = \alpha\bar{T} + (1 - \alpha)E$, where $0 \leq \alpha \leq 1$.

Finally, based on the $\bar{\bar{T}}$ matrix, the *Power method* is applied, calculating the stationary distribution matrix. Any of its columns yields the stationary vector, whose elements represent the global probability of accessing or manipulating a given domain class field of the application.

5 Results and Evaluation of the System

To evaluate the effectiveness of each method for predicting data access patterns in object-oriented applications, we resorted to two different benchmarks. The first benchmark was the TPC-W, which was originally presented by Smith [32] and specifies an e- commerce workload that simulates the activities of a retail store website. Emulated users can browse and order products from the website.

The second of the benchmarks is the oo7, firstly presented by Carey [7]. It is often used to assess the performance of object-oriented persistence mechanisms. It strives to present a broad set of operations, allowing for the building of a comprehensive performance profile. The oo7 has been designed to boast properties common to different CAD/CAM/CASE applications, although in its details it does not model any specific application. The evaluation is performed through the execution of a series of traversals, updates, and query operations over the underlying object model.

Due to the fact that the results from the two benchmarks are very similar, both in terms of the precision of the predictions generated by the models as well as with regards to the performance overheads due to the gathering of statistical information, only the results obtained from the oo7 benchmark shall be presented and discussed here.

There are some random behavioural elements in the oo7 benchmark, but these are not sufficient to demonstrate the validity of a system that aims to predict the behaviour of a target application. Accounting for that, Monte Carlo simulations [3] are employed, making the oo7 benchmark behave like a stochastic process. A stochastic process is one whose behaviour is non-deterministic in that a system's subsequent state is determined both by the process's predictable actions and by a random element. This is done by modelling the number of invocations of the methods that compose the benchmark. A triangular distribution is used to perform the modelling. This type of distribution is chosen because it is the one most commonly employed when dealing with a system about which there is little or none information regarding its behaviour. Three different triangular distributions are generated to model the benchmark invocations, namely with left, middle and right mode locations.

5.1 Bayesian Inference

As described in Section 4.1, the Bayesian inference technique uses two sources of data to make a prediction. One of them is the *previous*, and the second one is the *current*. The *current* data is used to update the *previous* collected information, by making it reflect the patterns that have been most recently observed.

Based on the Bayesian method, both the physical uncertainty related to the considered variable as well as the statistical uncertainty related to the model parameters can be quantified. An important property of the Bayesian analysis is that the uncertainty of the prediction is reduced. This may be seen in Figure 2 (up).

The x-axis of the histogram (Figure 2, down) represents the index of the probability classes within which a field belongs. The index i is defined by $i = floor[(p + 0.1)/0.1]$ where $p \in [0, 1]$. It should be noticed that, whereas the bars in the previous and current histograms indicate the number of fields whose observed probability of being read/written belongs to a given class, the predicted histogram reflects the fluctuation in the relative importance of each of the classes, which is expected to be seen in the following executions of the application (see textbook on Bayesian statistics, e.g. [5] and [26]).

As these results show, the great majority of the application fields belong to low probability classes, while there are only a few, which present the maximum probability of being accessed. This field distribution simplifies the process of making decisions with regards to optimizations. This is because most of the fields are not likely to be accessed and, as a result of that, the set of fields and associated classes, which need to be dealt with, is compact and easier to handle.

To evaluate the precision of the results generated by the system, it is necessary to compare the predictions generated during a given run of the benchmark with the actual patterns observed during its following run. To check whether the mean of the prediction is significantly different from the one observed in the next execution, we resorted to a null hypothesis test.

The null hypothesis states that the mean value of the prediction is not significantly different from the mean value of the patterns that were actually observed during the subsequent execution.

The Z test statistic for this type of check can be found in Box [5] as:

$$Z = \mu_1 - \mu_2 / \sqrt{\sigma_1^2/n_1 + \sigma_2^2/n_2} \qquad (3)$$

where μ_1 and μ_2 are the mean values, σ_1^2 and σ_2^2 are the variances and n_1 and n_2 are the sample sizes, for the predicted and the observed, respectively. This is also known as the decision rule.

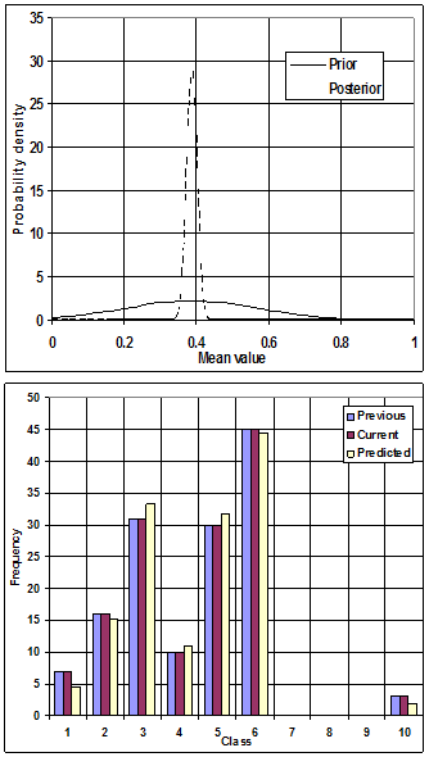

Figure 2: Bayesian inference, left mode location

The critical value for Z is defined as ± 1.96 , which corresponds to 0.05 level of significance to reject the null hypothesis, and equals to 95% of the area below the curve of the normal probability density function. It should be also noted that this particular value of 0.05 for the level of significance was chosen because it corresponds to the most common value used in engineering practice for such tests.

For the three different input modes tested here (left, middle and right), the calculated Z values ($Z_l = -0.33$, $Z_m = 0.34$ and $Z_r = 1.49$) belong to the area of 95% level of confidence. This means that at the 0.05 level of significance, there is no significant difference between the mean values of the predicted and subsequently observed (patterns). Consequently, we may conclude that the predictions created by the system are precise. They are able to indicate correctly the tendencies that are actually observed in the next executions of the application.

5.2 Importance Analysis

The results obtained by applying the Model based on Importance Analysis over the oo7 benchmark shall be presented in this section. Before the results themselves are shown, it is important to take under consideration the criticality rank table below.

Table 1: Criticality rank table

LAG	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

As has been explained in the theoretical section of the Model based on Importance Analysis, tables such as the one presented in Table 1 are used as guidelines for the decision making process of determining whether a given item should be considered as important or critical, with regards to the type of analysis being performed. In the context of the current work, such an item corresponds to the field of a given application class, whereas its importance or criticality is associated with the probability of that field being accessed by the application in a given context. Consequently, if, by using the RPN value of a given field, the position obtained belongs to the ones in the dark grey area, then the field is considered highly important for the operation of the application. If, on the other hand, the position of a field is within the non-shaded area, then it is very unlikely that it will be needed by the application. For the remaining fields in the intermediate area, their importance is deemed as average.

Passing on to the actual results, these may be observed in the histograms presented in Figure 3, where the left mode location is shown. The z-axis of this histogram corresponds to the number of fields whose RPN values equal the number obtained by multiplying their associated x and y values. The x-axis indicates the local probability access class to which a given field belongs,

while the y-axis is relative to the global probability access class of the fields.

A remark to be made is that most of the application fields belong to the lowest probability classes, while, at the same time, very few of them belong to the higher probability classes. This trend is similar to the one observed in the Bayesian Inference model results, and the benefits ensuing from it are the same, namely a smaller data set of likely to be needed data is easier to manage when trying to perform optimizations.

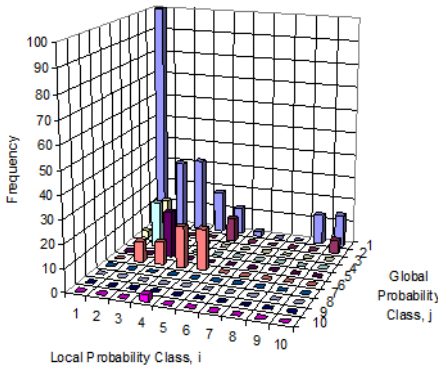

Figure 3: Criticality, left mode location

5.3 Markov Chains

The results generated when using the Markov Chain model over the oo7 benchmark shall be presented next. The two main outputs of this analysis technique are an access probability transition matrix and an access probability stationary vector. The matrix contains the observed probabilities of accessing a given field immediately after having accessed another field. The stationary vector presents the global access probabilities, which, in the work presented here, is calculated based on the *Power Method*.

The results achieved from executing the left mode location of the oo7 benchmark may be observed in Figure 4 and 5. Figure 4 presents the access probability matrix for the left mode location. The value present in a cell with coordinates (i, j) corresponds to the probability of accessing field i immediately after field j has been accessed. It should be noted that the diagonal formed from the lower left corner, towards the upper right corner, results from consecutive accesses to the same field, which correspond, most likely, to iterative access patterns.

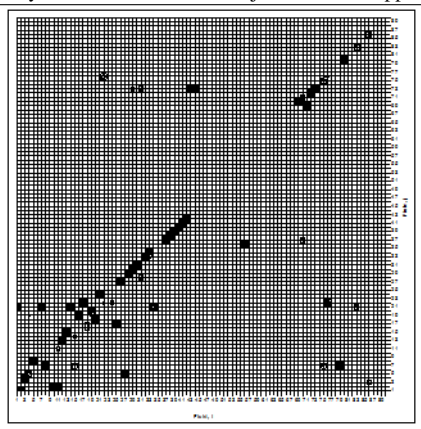

Figure 4: Access probability transition matrix, left mode

Due to the fact that the matrix generated has a size of 90×90, which corresponds to the number of fields present in the classes of the benchmark, it is not possible to present the numerical values of the cells of the matrixes. As such, the format employed only shows precisely where the probability is greater than zero and gives an indication of how high that probability is based on the degree of shading applied to the cell under consideration.

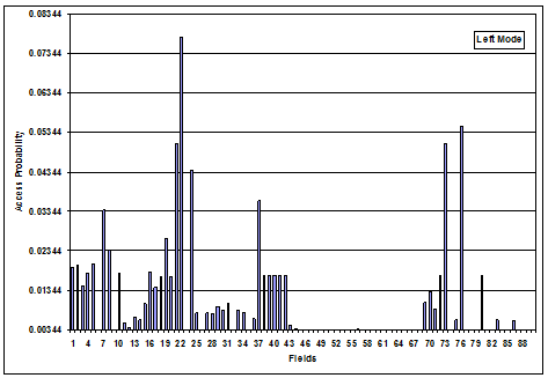

Figure 5: Access probability stationary vector, left mode location

Figure 5 shows the access probability stationary vector for the left mode location. The x-axis of the histogram corresponds to the field identificators, whereas the y-axis indicates the global probability of that field being accessed by the application. The y-axis is normalized, and, as such, all values belong to the interval $[0, 1]$. It is important to note that the minimum of the histograms, along the y-axis, is adjusted so that the bars of any field, whose probability of being accessed equals 0.00344, are not displayed. This is done with the aim of presenting in a clearer way the fields, which

have been effectively accessed during the execution of the benchmark. As a result of the application of the *Random Surfer Model*, any field that has never been accessed ends up with a minimal residual probability of being accessed, which, in the current situation, equals 0.00344.

It should be noted that the trend observed in the results of the other two models, with regards to the general distribution of fields (and associated classes) within the probability classes is preserved here. In other words, most of the fields have very low probability of being accessed, while the ones with the highest probability form a small and compact set.

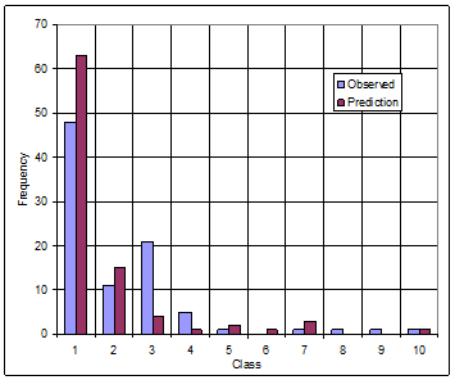

Figure 6: Populations used for Z value test, left mode

The null hypothesis test employed in Section 5.1 is employed here, once again, with the aim of evaluating the results generated by this model. The null hypothesis is formulated in the same way as the one used for the Bayesian Inference model, and, as such, the prediction generated by the system, under the form of the stationary vector with the field access probabilities, is compared to the actual access probabilities observed during a subsequent execution of the benchmark, modelled with the same location mode invocations. The data on which the test is based can be seen in Figure 6, where the left mode location is presented. The **Z** values obtained for the three mode location invocations of the benchmark are all practically equal to zero (belong to the area of 95% level of confidence). This leads us to the conclusion that the predictions generated by the system, while employing the Markov Chain model, are very precise. They are able to indicate correctly the actually observed tendencies in future executions of the target application.

6 Model Evaluation

It is possible to perform a comparison in terms of the properties presented by each of the three distinct models developed in this work. With regards to the semantic complexity presented by these three models, the one based on Bayesian Inference is clearly the most intricate, followed by the one based on Markov Chains, and lastly, the simplest of the three is the one based on Importance Analysis. It should be noted that the scale for this comparison is relative, and, as such, even the "simplest" of the models is far from trivial in practice.

The stability of the output produced by each of the models, in function of the fluctuations observed in the behaviour of the target application is considered next. It has been verified that the most stable of all models is the Markov Chain, the one based on Importance Analysis presents an intermediate stability, whereas the one based on Bayesian Inference reveals itself the most prone to fluctuations, based on the variations observed in the application behaviour.

Finally, with regards to the results generated by the three models, it is very difficult to compare them, because the models employed are significantly different and only by using their outputs in some optimization technique, such as guided prefetch or caching policies could they be compared effectively. Regardless of this, it is possible to observe a common trend shared by the results of all three of the three new models developed here, with regards to the majority of fields belonging to the lowest probability classes. In all three models, only a few of the application fields belong to higher access probability classes. The most likely to be accessed fields are the same for all three models.

7 Overhead Analysis

There is a significant amount of code injected in the application to collect all the information needed to build the models. The need to execute this code causes overheads and penalizes the performance of the application, in comparison with its non-instrumented version. Consequently, it is important to measure those overheads to determine whether they are acceptable in the context of the normal operation of the application. Another aspect to be taken under consideration is the memory space needed for the storage of the statistical data gathered up to a given point in time.

The mechanisms for gathering the input data for the Importance and Bayesian analysis models are significantly different from those employed for the Markov chain analysis, and, as such, their respective overheads shall be taken under consideration separately. Addition-

ally, it should be noted that whereas the data acquisition mechanisms for the former two analysis models are expected to operate without interruptions, while the application is operating normally, the gathering of data for the Markov chain analysis takes place only during the training period of the model. Consequently, the restrictions regarding the overheads introduced by the Markov chain data collection are much less strict than the ones for the other two models, since they are expected to keep gathering their data during the normal execution of the application.

The overhead analysis for the first two models (Bayesian and Importance Analysis) is presented next. The weighted average, $\overline{overhead}$, is calculated as:

$$\overline{overhead} = \frac{\sum_{i=1}^{n} overhead_{method,i} \times t_{method,i}}{\sum_{i=1}^{n} t_{method,i}} \quad (4)$$

where n is total number of methods, $overhead_{method,i}$ is the overhead, in percentage, associated with method with index i and $t_{method,i}$ is the execution time of method indexed by i. The weighted average of the performance overhead equals 5.15%. As a result of that, the instrumented version, for the Bayesian and Importance Analysis models is, on average, about 5% slower, in its execution, when compared with the original one. It is deemed that this performance penalty is acceptable.

With regards to the analysis of overheads introduced by data gathering for the Markov Chain model, the overhead weighted average, calculated by Eqn (4), for this case is 9.14%. Consequently, whenever the target application is in training mode for the acquisition of statistical data for the Markov Chain based model, it will be performing, on average, 9% slower than its non-instrumented version.

With regards to the additional memory requirements, due to the storage of the statistical data of observed access patterns, it is not possible to verify any "observable" difference in the memory needed by the original benchmark and the one employing the system developed here. An actual example would be to say that the statistical data gathered from several executions of the benchmark did not exceed several hundred KB, when saved on the hard disk, while the benchmark process would require a couple of hundred MB of memory, when executing. This allows us to conclude that the memory requirements for the storage of the statistical data necessary for the generation of predictions by the system are negligible.

8 Conclusions

The work presented here developed a new system for analyzing and predicting the most probable access patterns performed over domain data during the execution of object-oriented applications. The correct and accurate functioning of the system was established through the execution of the TPC-W and oo7 benchmarks. The three stochastic models implemented with the system are Bayesian Inference, Importance Analysis, and Markov Chains. Through them, predictions about the data access patterns, which are most likely to be performed by target applications during their execution, are generated.

Several distinct benchmark cases were simulated and analysed. The results were shown to be satisfactory for each of the models employed. The statistical data used as input for the models is acquired through code that is injected automatically into the target applications at compile-time. The overheads introduced into the target applications, associated with the data gathering process, are acceptable and range from 5% for the Bayesian Inference and Importance Analysis models to 9% for the Markov Chains.

The methodology developed with this work has been demonstrated to be flexible enough to be applied to any object-oriented application. It is not necessary to make any modifications whatsoever to the new system developed here in order to apply it to different target applications.

9 Acknowledgment

This work was partially supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and by the Specific Targeted Research Project (STReP) Cloud-TM, which is co-financed by the European Commission through the contract no. 257784. The first author has been funded by the Portuguese FCT (Fundação para a Ciência e a Tecnologia) under contract SFRH/BD/64379/2009.

References

[1] Adali, S., Candan, K. S., Papakonstantinou, Y., and Subrahmanian, V. S. Query caching and optimization in distributed mediator systems. *SIGMOD Rec.*, 25:137–146, June 1996.

[2] Aitchison, J. and Dunsmore, I. *Statistical Prediction Analysis*. Cambridge University Press, New York, NY, USA, 1975.

[3] Berg, B. *Markov Chain Monte Carlo Simulations and Their Statistical Analysis*. World Scientific, 2004.

[4] Bernstein, P. A., Pal, S., and Shutt, D. Context-based prefetch an optimization for implementing objects on relations. *The VLDB Journal*, 9:177–189, December 2000.

[5] Box, G. and Tiao, G. *Bayesian Inference in Statistical Analysis*. Wiley, New York, NY, USA, 1992.

[6] Brin, S. and Page, L. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30:107–117, April 1998.

[7] Carey, M., Dewitt, D., and Naughton, J. The oo7 benchmark. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 12–21, 1993.

[8] Dar, S., Franklin, M. J., Jónsson, B. T., Srivastava, D., and Tan, M. Semantic data caching and replacement. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 330–341, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[9] DoD, U. S. Procedures for performing a failure mode effects and criticality analysis. MIL-STD-1629A, 1984.

[10] Drapeau, S., Roncancio, C., and Guerrero, E. Generating association rules for prefetching. In *Proceedings of the ICDCS Workshop of Knowledge Discovery and Data Mining in the World-Wide Web*, pages F15–F22, Taipei, Taiwan, 2000.

[11] Garbatov, S. and Cachopo, J. Importance analysis for predicting data access behaviour in object-oriented applications. *Journal of Computer Science and Technologies*, 14:37–43, 2010.

[12] Garbatov, S. and Cachopo, J. Predicting data access patterns in object-oriented applications based on markov chains. In *Proceedings of the 2010 Fifth International Conference on Software Engineering Advances*, ICSEA '10, pages 465–470, Washington, DC, USA, 2010. IEEE Computer Society.

[13] Gerlhof, C. A. and Kemper, A. A multi-threaded architecture for prefetching in object bases. In *Proceedings of the 4th international conference on extending database technology: Advances in database technology*, EDBT '94, pages 351–364, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[14] Han, W., Moon, Y., and Whang, K. Prefetchguide: capturing navigational access patterns for prefetching in client/server object-oriented/object-relational dbmss. *Inf. Sci.*, 152:47–61, June 2003.

[15] Hsu, W. W., Smith, A. J., and Young, H. C. I/o reference behavior of production database workloads and the tpc benchmarks - an analysis at the logical level. *ACM Trans. Database Syst.*, 26:96–143, March 2001.

[16] Ibrahim, A. and Cook, W. Automatic prefetching by traversal profiling in object persistence architectures. In *Proceedings of the European Conference on Object-Oriented Programming*, ECOOP '06, pages 50–73, 2006.

[17] Kapitskaia, O., Ng, R. T., and Srivastava, D. Evolution and revolutions in ldap directory caches. In *Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '00, pages 202–216, London, UK, 2000. Springer-Verlag.

[18] Kececioglu, D. *Reliability Engineering Handbook*. Prentice-Hall Inc, New Jersey, NJ, USA, 1991.

[19] Knafla, N. *Prefetching Techniques for Client/Server, Object-Oriented Database Systems*. PhD thesis, Citeseer, 1999.

[20] Kroeger, T. M., Long, D. D. E., and Mogul, J. C. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, pages 2–2, Berkeley, CA, USA, 1997. USENIX Association.

[21] Langford, J. W. *Logistics: Principles and Applications*. McGraw Hill, 1995.

[22] Lei, H. and Duchamp, D. An analytical approach to file prefetching. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 21–21, Berkeley, CA, USA, 1997. USENIX Association.

[23] Li, M. and Li, J. On the predictability of long-range dependent series. *Mathematical Problems in Engineering*, 2010:1–9, 2010.

[24] Li, M. and Lim, S. Modeling network traffic using generalized cauchy process. *Physica A: Statistical Mechanics and its Applications*, 387(11):2584–2594, 2008.

[25] Li, M. and Zhao, W. Representation of a stochastic traffic bound. *IEEE Trans. Parallel Distrib. Syst.*, 21:1368–1372, September 2010.

[26] Lindley, D. *Introduction to Probability and Statistics from a Bayesian Viewpoint*. Cambridge University Press, 1976.

[27] Liskov, B., Adya, A., Castro, M., Ghemawat, S., Gruber, R., Maheshwari, U., Myers, A. C., Day, M., and Shrira, L. Safe and efficient sharing of persistent objects in thor. *SIGMOD Rec.*, 25:318–329, June 1996.

[28] Luo, Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Woo, H., Lindsay, B. G., and Naughton, J. F. Middle-tier database caching for e-business. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 600–611, New York, NY, USA, 2002. ACM.

[29] Madhyastha, T. M. and Reed, D. A. Input/output access pattern classification using hidden markov models. In *Proceedings of the fifth workshop on I/O in parallel and distributed systems*, IOPADS '97, pages 57–67, New York, NY, USA, 1997. ACM.

[30] Meyn, S. and Tweedie, R. L. *Markov Chains and Stochastic Stability*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

[31] Palmer, M. and Zdonik, S. *Fido: A cache that learns to fetch*. Brown University, Dept. of Computer Science, 1991.

[32] Smith, W. Tpc-w: Benchmarking an ecommerce solution. *White paper, Transaction Processing Performance Council*, 2000.

[33] Wang, D. and Xie, J. An approach toward web caching and prefetching for database management system. In *SIGMOD*. Citeseer, 2001.

[34] Whaley, J. A portable sampling-based profiler for java virtual machines. In *Proceedings of the ACM 2000 conference on Java Grande*, JAVA '00, pages 78–87, New York, NY, USA, 2000. ACM.

[35] Wiedermann, B. and Cook, W. R. Extracting queries by static analysis of transparent persistence. *SIGPLAN Not.*, 42:199–210, January 2007.