Predicting Data Access Patterns in Object-Oriented Applications based on Markov Chains

Stoyan Garbatov and João Cachopo
Software Engineering Group
Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento, INESC-id
Lisbon, Portugal
stoyangarbatov@gmail.com and joao.cachopo@inesc-id.pt

***Abstract* — This work aims to create an innovative system for analyzing and predicting the behaviour of object-oriented applications, with respect to the domain objects they manipulate, based on Markov Chains. 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. The results are precise, regarding the observed behaviour and the overheads introduced by the data acquisition are low. The system is sufficiently flexible to be applied to a broad spectrum of object-oriented applications.**

Keywords-data access; Markov Chains; Monte Carlo

I. INTRODUCTION

Most applications need to fetch data from external resources to perform their functions. This is a time consuming operation. Because of this, applications that access large volumes of data must be carefully engineered to have acceptable performance.

The cost of fetching data from an external resource depends on a series of factors. A good design rule is to minimize the number of round-trips made by the application to load the data. On the other hand, it should be also sought out to minimize the amount of data fetched. Consequently, performing a single round-trip that fetches exactly the data that is needed for a given operation would be the optimal solution. A common example of such data-fetching approach is the use of complex SQL queries to retrieve data from multiple tables at once from a relational database. Unfortunately, this idealized goal is seldom possible to achieve. The reasons are manifold.

The current state-of-the-practice is to leave to the programmers the burden of knowing what to fetch and when to do it. However, as applications become more complex and are developed with higher-level languages, knowing beforehand which data is needed for a particular computation, becomes humanly unfeasible.

Moreover, even if the data-fetching patterns for an application are fine-tuned, these adjustments may become useless or even counterproductive when either the requirements for the application or the data structure change.

Additionally, many applications use caches to reduce the need to fetch data from external resources, and, thereby, accelerate the application. The use of these caches and their dynamic contents influence the optimal fetching strategy.

The data-fetching strategy for an application should not be a responsibility of the programmers. It should be determined automatically, based on the data-access behaviour observed.

Several related works have been published. Knafla [1] presented a methodology where the main goal is to facilitate the prefetching of persistent objects through the prediction of data access. The relationships between the objects are modelled by a discrete-time Markov Chain. The work offers an interesting view regarding the use of stochastic models for predicting the behaviour of applications. However, only the actually existing relationships between domain objects are taken under consideration with the model. It may be more interesting to consider the effectively observed data accesses, instead of only the static domain object hierarchy.

Bernstein et al. [2] presented a technique for data prefetching. The main concept behind it is to associate a context to every object at the time it is loaded. This subsequently allows using the context of the object and the concrete portion of an object's state that is loaded to interpolate about the probability of the application needing to manipulate the same portion of state of other objects sharing that context. That enables the loading of data that would be needed by the application in a pre-emptive fashion (prefetching it), effectively reducing the time lost while waiting for the data to be loaded on demand. Other works related to data accesses and manipulations [3-9], loading and prefetching [10-18] and caching can be seen in [19-28].

The work presented here is a continuation of the one developed in [29], where the analysis is based on the Bayesian inference model, and is also a part of [30] and [31].

The system developed here aims to analyse and predict when an application needs to manipulate its domain objects. The domain objects contain data needed for the proper application operation. The data originates from external sources, like data-bases, but the domain objects wrap this data. The analysis considers how these objects are manipulated, with the goal of improving the application performance. The use of the results for optimizations is not a part of the scope of this work. The term "data" shall be used to designate the application domain objects.

This article describes the design and implementation of a system that predicts the data-access patterns for a Java

application. It captures all of the data accesses made by the application during its execution, and uses Markov Chains analysis over the collected data to predict future data accesses. This data reflects not just the static domain model hierarchy but the way the application actually navigates it, based on the effectively observed sequences of data accesses.

The system was designed to integrate seamlessly with the development of a Java application, and requires minimal effort from the programmer. Additionally, the system is efficient both in the amount of memory that it requires and in the performance overheads that it introduces.

II. System Description

The system is composed of three modules: a code injection module, a data acquisition module, and a data analysis module. The first module is responsible for injecting automatically code into the target applications. This code is necessary for the invocation of 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 one – the data acquisition module – deals with collecting data about the actual behaviour of the target application, with regards to the data accesses that are performed. Finally, the data analysis module is responsible for applying the discrete time Markov Chain model 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 data it will access.

A. 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. To capture these patterns, it is necessary to identify all the points where domain objects are accessed within the application and instrument them.

The system performs this instrumentation at compile-time by means of bytecode 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 (http://www.csg.is.titech.ac.jp/~chiba/javassist/) library for code injection.

The first instrumentation phase injects 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 work presented here, the context corresponds to the method in which the data accesses take place, but it can be defined in a different way, if the situation so demands (for instance, the context can correspond to the execution of a given service or even a sequence of method invocations that precede the current point of execution).

To identify *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.

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 the domain classes of the application. These methods resolve the surrounding *context* within which the field is accessed and update the associated statistical information. This is done according to the type of access: a distinct method is invoked if the field is read or written.

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 further ado.

An important aspect of the solution presented here is the data structures used to store the statistical data. During the instrumentation phases, the system performs an analysis of the domain hierarchy 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 domain classes. Each of these *PClass* instances contains a set of *PField* instances, which are used to represent each of the existing fields. The information kept covers not only the name and type of a field, but also the number of times that it has been accessed in a *context*. This information is later used for the probabilistic analysis.

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

It is possible to estimate the upper bounds on the memory requirements for maintaining these structures. The memory is proportional to the number of domain classes, times their average number of fields, times the average number of distinct *contexts* where each class is accessed during the execution. Additionally, memory requirements do not grow continuously, independently of the duration of the execution of the application. Overall, memory consumption is negligible when compared to the application memory.

Given that the probabilistic analysis iterates through the *PField* and *PClass* instances, time spent on it is also proportional to the previously stated bound.

B. Model Implementation

The first phase of the Markov Chain model [32] builds a transition matrix. This matrix contains the probabilities of transiting from one system state to another (automata theory). One of the main goals of this work is to allow the identification and prediction of access patterns performed by applications. The states correspond to the manipulation of a given domain class field. In the transition matrix $\mathbf{T} = [t_{ij}]$, the cell t_{ij} contains the probability of manipulating field i immediately after having manipulated field j.

To calculate these probabilities, each of the identified *contexts* keeps a hash-table containing the statistical data

about the sequences of observed field accesses. The hash-table keys correspond to *PFields*, whereas the associated values are sets of *PFields*. These *PField* sets contain access information for the situations when they have been accessed immediately after the hash-map key (*PField*) to which they are associated.

All the necessary input for the analysis is collected during a training period, which is to be performed only once, as long as it is representative of the service life of the application. During this period, whenever there is a field access, the surrounding context is determined. Next, the last field that has been accessed in that *context* is identified and used to index the *context's* hash-map 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 accumulation of data about what fields are accessed after a given one and the number of times this has been observed.

When the collected statistical data is representative, the transition matrix can be built. The first phase consists in creating a square matrix $\mathbf{T}$ whose size is $n \times n$, 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 $\mathbf{T}$, each of its $i \in [1,n]$ cells contains the number of times that the field i has been accessed immediately after j. After this, a normalization phase is performed. 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, $\overline{\mathbf{T}}$.

This matrix does not present all the necessary properties in order to be suitable for the *Power method* [33]. This method consists in calculating the powers of the transition matrix to obtain the stationary distribution.

The *Random Surfer Model* [34] is applied to make the matrix suitable. Each of the columns of the newly formed matrix are iterated through, determining if there one whose elements are all zero. If this is so, then the elements of the given column j are set to $t_{ij} = 1/n$.

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

Finally, based on the $\overline{\overline{\mathbf{T}}}$ 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 field in the application.

III. Results and Evaluation of the System

One of the benchmarks used for appraising this system's behaviour is the TPC-W. It has been presented by Smith in [35] 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 main evaluation metric is the WIPS – web interactions per second that can be sustained by the system under test.

The second of the benchmarks is the oo7, firstly presented by Carey et al. [36]. 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. The performance metric used is the time these operations take to execute.

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 [37] 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.

The results generated when using the Markov Chain model shall be discussed here. The conclusions derived from both benchmarks are very similar, and because of that the TPC-W results are omitted. The results of all three mode locations of the oo7 are distinct, but they are consistent in the conclusions to be obtained from them. To demonstrate the system capability, only the results of the left invocation mode location are presented.

The two outputs of the Markov Chain analysis 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 this work, are based on the Power Method.

The results achieved from executing the oo7 benchmark may be observed in Fig. 1. It shows the access probability stationary vector. The *x*-axis of the histogram corresponds to the application domain class field identificators, whereas the *y*-axis indicates the global probability of that field being accessed. The *y*-axis is normalized - all values belong to the interval [0,1]. 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.

An interesting trend that was observed is the fact that the great majority of the fields have a very low probability of being accessed, whereas the ones with the highest probability form a small and compact set.

To evaluate the precision of the model predictions, a null hypothesis [38] is employed here. The null hypothesis states that the mean value of the prediction is not significantly different from the mean value of the behaviour that is actually observed during a subsequent execution. The prediction generated by the system, under the form of the stationary vector, 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 Fig. 2. The Z values obtained for the three mode location invocations of the benchmark are practically equal to zero. Consequently, all three Z values belong to the area of 95% level of confidence. This leads us to the conclusion that the predictions generated by the system, when 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.

There is a significant amount of code injected in the application to collect all the information needed to build the model. 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 training period of

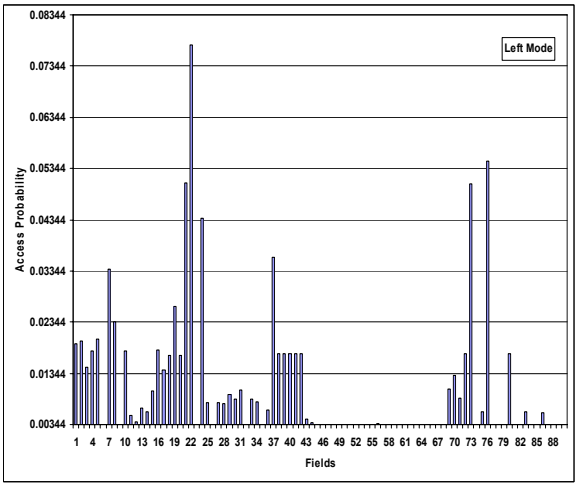

Figure 1. Access probability stationary vector.

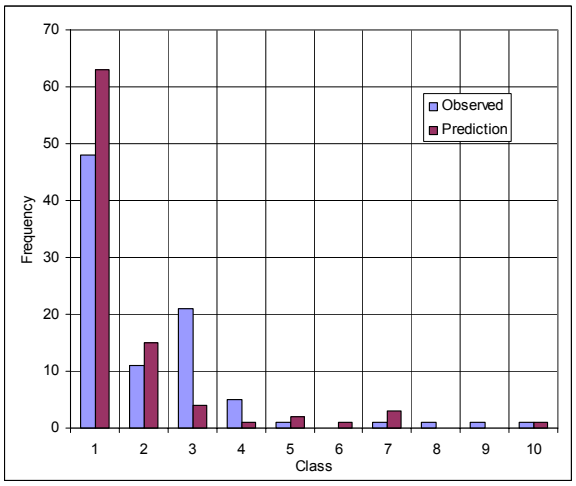

Figure 2. Populations used for Z value test.

the system. 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 overhead analysis for the Markov Chain model is presented next. The execution times of the main methods of the oo7 benchmark are summarized in Fig. 3.

In the *x*-axis of the histogram presented in Fig. 3, we have each of the 14 methods that define the oo7 benchmark. For each of these methods there are two bars: the right one indicates the execution time (in milliseconds) of the method in the original benchmark and the left one is the execution time of the instrumented version. The weighted average is $\overline{overhead} = \sum_{i=1}^{n} overhead_{method,i} \cdot t_{method,i} \Big/ \sum_{i=1}^{n} t_{method,i}$, 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 overhead is 9.14%. Whenever the target application is in training mode for the acquisition of statistical data for the 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. A concrete example would be to say that the statistical data gathered from several executions of the benchmark did not exceed a couple of hundred KB, when saved on the hard disk, while the benchmark process required several hundred MB of memory when executing. It can be concluded that the memory requirement for the storage of the statistical data is negligible.

IV. TYPE VERSUS INSTANCE ANALYSIS

It is possible to distinguish two types of approaches, based on their granularity, during the analysis of the behaviour of an application, with regards to the data

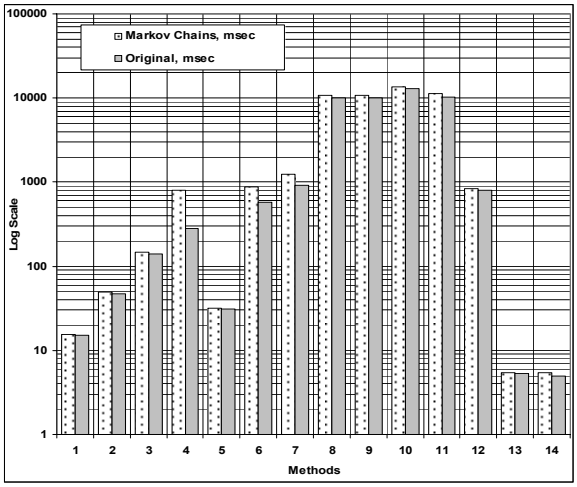

Figure 3. Method execution times (msec)

manipulations it performs. The first approach corresponds to an analysis based on the individual objects (instances) observed to be accessed while the second one is based solely on the types of the objects being accessed. The system presented here adheres to the analysis based on the type of the objects The properties boasted by both approaches shall be addressed, justifying the choice made here.

Bowman and Salem [17] point out that statistical prefetching techniques, which are based on information about the individual object instances being accessed, need storage proportional to the number of distinct objects, for providing the best possible predictions. Furthermore, they rely on an expensive training period. It is also the case that patterns based on object identifiers do not generalize to similar patterns that operate over different objects. Prediction techniques based on individual object instances are not very capable of extracting the general access pattern which lies under the specific accesses being observed.

The analysis techniques based on object types are not demanding, with regards to the space needed to keep the acquired behavioural data. Additionally, they can identify the underlying general access patterns effortlessly, assuming that the models employed for their functioning are well implemented. The time necessary for the acquisition of data for a stable analysis is shorter when compared to that of the other family of techniques. A disadvantage present in the use of these techniques may reside in the fact that, once the predictions are available, it may be difficult to identify the specific object instances over which to apply whatever optimization action is intended.

V. CONCLUSIONS

This paper presented a new system for analyzing the behaviour of target applications with regards to the data accesses they perform. The data accesses are analyzed and predicted using advanced probabilistic techniques employing Markov Chains. Several scenarios were generated and executed, and the subsequent results were analyzed. The system developed for data access behaviour analysis was applied over the TPC-W and oo7 benchmarks. The oo7 was randomized through the use of Monte Carlo simulation.

The overheads introduced by the system were shown to be in the order of 9%, when comparing the instrumented application with the original one. The predictions were demonstrated to be very precise.

The system developed here is flexible enough to be used as a basis for a set of optimization techniques. Special benefits may be obtained when the data being analyzed is persistent. This should allow for the application of optimizations regarding the way that the data is loaded, stored, or cached.

ACKNOWLEDGMENT

This work has been performed within the Pastramy project (PTDC/EIA/72405/2006), funded by the Portuguese FCT (Fundação para a Ciência e a Tecnologia). The first author has been funded by the FCT under contract SFRH/BD/64379/2009.

REFERENCES

[1] Knafla, N. Analysing object relationships to predict page access for prefetching. in Eighth Int. Workshop on Persistent Object Systems: Design, Implementation and Use (POS-8), 1998.

[2] Bernstein, P., S. Pal, and D. Shutt. Context-based prefetch for implementing objects on relations. in 25th Very Large Data Base Conference (VLDB99), 1999.

[3] Willis, D., D. Pearce, and J. Noble. Efficient object querying for java. in European Conference on Object-Oriented Programming (ECOOP), 2006.

[4] Wiedermann, B. and W. Cook. Extracting queries by static analysis of transparent persistence. in 34th Symposium on Principles of Programming Languages (POPL07), 2007.

[5] Wiedermann, B., A. Ibrahim, and W. Cook. Interprocedural query extraction for transparent persistence. in Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA08), 2008.

[6] Madhyastha, T. and D. Reed. Input/output access pattern classification using hidden Markov models. in Fifth Workshop on Input/Output in Parallel and Distributed Systems, 1997.

[7] Liskov, B., et al. Safe and efficient sharing of persistent objects in Thor. in ACM SIGMOD International Conference on Management of data, 1996.

[8] Hsu, W., A. Smith, and H. Young, I/O reference behavior of production database workloads and the TPC benchmarks: An analysis at the logical level. ACM Transactions on Database Systems. 26(1): p. 96-143, 2001.

[9] Florescu, D., et al. Optimization of run-time management of data intensive web sites. in 25th International Conference on Very Large Data Bases. Edinburgh, Scotland, 1999.

[10] Lei, H. and D. Duchamp. An analytical approach to file prefetching. in USENIX Conference, 1997.

[11] Knafla, N., Prefetching techniques for client/server, object-oriented database systems. 1999.

[12] Ibrahim, A. and W. Cook. Automatic prefetching by traversal profiling in object persistence architectures. in European Conference on Object-Oriented Programming (ECOOP), 2006.

[13] Han, W., Y. Moon, and K. Whang, Prefetchguide: capturing navigational access patterns for prefetching in client/server object-oriented/object-relational dbmss. Information Sciences: An International Journal. 152(1): p. 47–61, 2003.

[14] Han, W., K. Whang, and Y. Moon. Prefetching based on the type-level access pattern in object-relational dbmss. in 25th Very Large Data Base Conference (VLDB99), 1999.

[15] Gerlhof, C. and A. Kemper. A multi-threaded architecture for prefetching in object bases. in 4th International Conference on Extending Database Technology, 1994.

[16] Drapeau, S., C. Roncancio, and E. Guerrero. Generating association rules for prefetching. in ICDCS Workshop of Knowledge Discovery and Data Mining in the World-Wide Web. Taipei, Taiwan, 2000.

[17] Bowman, I. and K. Salem. Optimization of query streams using semantic prefetching. in ACM SIGMOD Conference on Management of Data. Paris, 2004.

[18] Bernstein, P., S. Pal, and D. Shutt, Context-based prefetch - an optimization for implementing objects on relations. VLDB Journal: Very Large Data Bases. p. 177-189, 2000.

[19] Wang, D. and J. Xie, An approach toward Web caching and prefetching for database management systems. 2001.

[20] Palmer, M. and S. Zdonik. Fido: A cache that learns to fetch. in 17th International Conference on Very Large Data Bases. Barcelona, Spain, 1991.

[21] Luo, Q., et al. Middle-tier database caching for e-business. in ACM SIGMOD Conference on Management of Data, 2002.

[22] Kroeger, T., D. Long, and J. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. in USENIX Symposium on Internet Technologies and Systems, 1997.

[23] Keller, A. and J. Basu, A predicate-based caching scheme for client-server database architectures. VLDB Journal: Very Large Data Bases. 5(1): p. 35-47, 1996.

[24] Kapitskaia, O., N. Raymond, and D. Srivastava. Evolution and revolutions in LDAP directory caches. in 7th International Conference on Extending Database Technology. Konstanz, Germany: Springer, 2000.

[25] Haas, L., D. Kossmann, and I. Ursu. Loading a cache with query results. in 25th International Conference on Very Large Data Bases, 1999.

[26] Dar, S., et al. Semantic data caching and replacement. in 22th International Conference on Very Large Data Bases, 1996.

[27] Adali, S., et al. Query caching and optimization in distributed mediator systems. in ACM SIGMOD Conference on Management of Data, 1996.

[28] Willis, D., D. Pearce, and J. Noble. Caching and incrementalisation in the java query language. in Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA08), 2008.

[29] Garbatov, S., J. Cachopo, and J. Pereira. Data Access Pattern Analysis based on Bayesian Updating. in INForum 2009. Lisbon, 2009.

[30] Garbatov, S., J. Cachopo, and J. Pereira, Data Access Pattern Analysis based on Bayesian Updating. 2009, INESC-ID.

[31] Garbatov, S., Data Access Patterns Analysis and Prediction for Object-Oriented Applications, in Computer Science. 2009, Technical University of Lisbon, Instituto Superior Técnico: Lisbon.

[32] Meyn, S.P. and R.L. Tweedie, Markov Chains and Stochastic Stability: Cambridge University Press, 2005.

[33] Aitchison, J. and I. Dunsmore, Statistical Prediction Analysis. Cambridge: Cambridge University Press, 1975.

[34] Brin, S. and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. in Seventh International World Wide Web Conference: Elsevier B.V., 1998.

[35] Smith, W., TPC-W: Benchmarking an Ecommerce Solution, Intel Corporation, 2000.

[36] Carey, M., D. Dewitt, and J. Naughton. The OO7 Benchmark. in ACM SIGMOD International Conference on Management of Data, 1993.

[37] Berg, B., Markov Chain Monte Carlo Simulations and Their Statistical Analysis: Hackensack, NJ: World Scientific, 2004.

[38] Box, G. and G. Tiao, Bayesian Inference in Statistical Analysis. New York: Wiley, 1992.