Control and Cybernetics

vol. **38** (2009) No. 1

Predicting access to materialized methods by means of hidden Markov model*†

by

Mariusz Masewicz, Witold Andrzejewski, Robert Wrembel and Zbyszko Królikowski

Institute of Computing Science, Poznań University of Technology
Poznań, Poland
e-mail: {mmasewicz,wandrzejewski,rwrembel,zkrolikowski}@cs.put.poznan.pl

Abstract: Method materialization is a promising data access optimization technique for multiple applications, including, in particular object programming languages with persistence, object databases, distributed computing systems, object-relational data warehouses, multimedia data warehouses, and spatial data warehouses. A drawback of this technique is that the value of a materialized method becomes invalid when an object used for computing the value of the method is updated. As a consequence, a materialized value of the method has to be recomputed. The materialized value can be recomputed either immediately after updating the object or just before calling the method. The moment the method is recomputed bears a strong impact on the overall system performance. In this paper we propose a technique of predicting access to materialized methods and objects, for the purpose of selecting the most appropriate recomputation technique. The prediction technique is based on the Hidden Markov Model (HMM). The prediction technique was implemented and evaluated experimentally. Its performance characteristics were compared to: immediate recomputation, deferred recomputation, random recomputation, and to our previous prediction technique, called a PMAP.

Keywords: method materialization, hierarchical materialization, access prediction, Hidden Markov Model.

1. Introduction

Object technologies and systems (see Loomis, 1995) have been initially developed in order to support storing and processing complex data and in order

*This work was supported from the Polish Ministry of Science and Higher Education grant No. N N516 365834

†Submitted: June 2008; Accepted: October 2008.

to ease software development. The technologies have been successfully applied in such domains as, for instance, CAD, CAM, CASE, GIS (see Loomis and Chaudhri, 1998), multimedia applications, multiple programming languages, distributed computing systems like CORBA (see OMG, 2006), as well as in Internet applications.

Object technologies are based on an object data model (see Catell et al., 2000). In this model, real-world entities are represented by *objects*. Every object has its state and behavior. The *state* of an object is defined by the values of its properties that include attributes and relationships. A property of type relationship allows to relate one object to another object. The *behavior* of an object is defined by the set of *methods* (operations) that can be executed for an object, further called a *base object*. Objects are instances of classes. A *class* defines the common state and behavior of all its instances.

Methods can be very complex programs, they may access large number of objects, and their computation may last long. Therefore, efficient execution of methods has a great impact on system performance. Optimizing execution of methods is challenging, since methods are expressed in object languages, taking advantage of inheritance, polymorphism, overloading, and late binding. Moreover, source codes of methods are usually complex, with loops, conditional expressions, and calls to other methods.

A promising technique applied in optimization of method executions is called method materialization (also known as precomputation or caching). Basically, *method materialization* consists in: (1) computing the result of the first execution of a method, say m_i, for a given base object, say o_i, and with a given set of input argument values, and then (2) storing the result persistently on a disk. Every subsequent execution of m_i for the same object o_i and with the same set of input argument values can be handled by fetching the already materialized value.

Method materialization is a promising data access optimization technique not only in typical object systems but also in object-relational data warehouses (ORDWs). In ORDWs (see, e.g., Gopalkrishnan, Li and Karlapalem, 2000; Huynh, Mangisengi and Tjoa, 2000; Kandaswamy, 2003, and Konovalov, 2002), materialized object views (see e.g., Ali, Fernandes, and Paton, 2000; Czejdo et al., 2001, and Kuno and Rundensteiner, 1998) play an important role, similarly as in traditional data warehouses. Firstly, due to the expressive power of an object data model, object views are capable of accessing and transforming data of an arbitrary complex structure and behavior into a common model (see, e.g., Bukhres and Elmagarmid, 1995, and Fankhauser et al., 1998) used in ORDW. For example, by materializing methods, one is able to represent behavior of objects in terms of the relational data model. In this case, materialized methods are just seen as attributes having persistent values. Secondly, materialized object views may be used in query optimization. Moreover, in multimedia data warehouses (see, e.g., Arigon, Tchounikine and Maryvonne, 2006; Kim and Park, 2003, and Messaoud, Boussaid and Rabaséda, 2004) and spatial data warehouses (see, e.g., Bédard, Rivest and Proulx, 2006; Gorawski and Malczok,

2005; Gorawski and Kamiński, 2006, and Yu, Atluri and Adam, 2006), materialized methods are useful mechanisms for optimizing analytical processing of images, sounds, and spatial data.

A drawback of method materialization is that the materialized value of method m_i becomes invalid (outdated) when its base object is updated. As a consequence, the materialized value has to be recomputed (rematerialized). The materialized value of m_i can be recomputed either immediately, following the update of its base object (further called *immediate recomputation*) or just before reading the value of m_i (further called *deferred recomputation*).

On the one hand, immediate recomputation reduces response time for a user's application (session) since the method output is recomputed in advance. On the other hand, it may cause unnecessary recomputations, when after recomputing the value of m_i it is immediately invalidated by an update of the base object. On the contrary, the deferred recomputation better uses system resources since a materialized method is recomputed only when needed, but it causes delays in accessing invalidated methods since a user's application has to wait until a method is recomputed. For these reasons, there is a need for a technique that would support the selection of the most appropriate recomputation technique based on a current workload characteristic. Existing approaches to method materialization, i.e., Bertino (1991), Eder, Frank and Liebhart (1994), Jhingran (1991), Kemper, Kilger and Moerkotte(1994), Liu and Teitelbaum (1995), Liu, Stoller and Teitelbaum (1998) and Pugh and Teitelbaum (1989), do not address this problem.

In this paper we present a method recomputation technique, called *HMM recomputation*. It applies Hidden Markov Models (HMM) for the purpose of predicting access to materialized methods and their base objects. Based on the prediction, the most appropriate method recomputation technique is used, i.e. either the immediate or the deferred one. The HMM recomputation was implemented and evaluated experimentally. Its performance characteristics were compared to: the immediate recomputation, deferred recomputation, random recomputation, and to our previous recomputation technique, called PMAP (Masewicz et al., 2006).

The paper is organized as follows. Section 2 overviews related approaches to method materialization and maintenance. Section 3 presents basic concepts referred to in this paper, i.e. hierarchical materialization, the PMAP recomputation technique, and the Hidden Markov Model. Section 4 presents the concept of the HMM recomputation and Section 5 discusses its experimental evaluation. Finally, Section 6 summarizes and concludes the paper.

2. Related work

Several approaches to method materialization have been proposed in the research literature. The approaches can be characterized as: (1) supporting a persistent materialization, i.e., Bertino (1991), Jhingran (1991), Kemper Kilger

and Moerkotte (1994) and Liu and Teitelbaum (1995), and (2) supporting a temporal materialization, i.e., Eder, Frank and Liebhart (1994) and Pugh and Teitelbaum (1989).

The work presented in Jhingran (1991) analytically estimates costs of caching complex objects accessed procedurally. Two data representations are considered, i.e. a procedural representation and an object identity based representation. The maintenance of cached (materialized) values was not taken into consideration.

In Bertino (1991), the results of materialized methods are stored in a B-tree based index, called a *method-index*. While executing queries that use materialized method m_i, the system searches the method-index for the value of m_i before executing it. If the appropriate entry is found, then the already precomputed value is used. Otherwise, m_i is executed for an object. A method may be materialized provided that: it does not have input arguments, it computes values based on only atomic types, and it does not modify values of objects. Otherwise, a method is not materialized.

In Kemper, Kilger and Moerkotte (1994), the results of materialized methods are stored in the so-called *Reverse Reference Relation*. It contains an information on: an object used to materialize method m_i, the name of a materialized method, and the set of objects passed to m_i as arguments. For the purpose of method invalidations, every object has an appended set of identifiers of methods that used the object. Moreover, every materialized method m_i has an associated list of attributes, whose values were used for the materialization of m_i. In this approach, a system designer has to explicitly define in advance (during a system design phase) data structures for storing materialized results for all methods, but the defined data structures may never be used when methods are not materialized.

In Liu and Teitelbaum (1995), the authors proposed to decompose complex methods into the graph of component methods. The semantics of complex and component methods is then analyzed in order to figure out which results to cache. The approach requires huge secondary storage as method results are cached extensively. Moreover, the maintenance of cached results is not supported.

In Eder, Frank and Liebhart (1994), the concept of the so-called *inverse methods* was proposed. When an inverse method is used in a query, say Q, it is computed once, instead of computing it for each object returned by Q. The result of an inverse method is cached in memory only within the duration time of Q and it is accessible only to Q.

In Pugh and Teitelbaum (1989), the authors proposed to store results of method executions in a hash table stored in memory. In this technique, for the purpose of increasing the usage of cached results, complex functions are decomposed to simple ones, whose results are cached. The approach supports caching methods with constant input values only, i.e. various calls of the same method have to provide the same value of input arguments. This feature strongly limits the application of the approach.

Two concepts loosely related to method materialization concern: a cost model of method executions (see Gardarin, Sha and Tang, 1996) and indexing methods along an inheritance hierarchy (see Kratky et al., 2005). The cost model developed in Gardarin, Sha and Tang (1996) includes the number of O/I operations and CPU time, but it does not consider method materialization. In Kratky et al. (2005), the authors proposed and evaluated R-tree based indexes for the optimization of searching methods. This approach focuses on indexing metadata on methods, rather than method values.

The common limitation of the approaches mentioned above is that they do not support any technique for selecting the most appropriate method maintenance (rematerialization, recomputation). A wise method maintenance should be based on a technique for predicting operations that will appear in a system, as discussed in Section 1.

The research in the prediction area is conducted in multiple domains, from meteorology, e.g., weather and tornado forecasting (see Drton et al., 2003), genetics (see Deng and Ali, 2004), financial and stock markets (see Oral and Kettani, 1989), to computer science, e.g., CPU communication (see Kaxiras and Young, 2000), file access (see Pâris, Amer and Long, 2003), Web user behavior (see Dongshan and Junyi, 2002), intrusion detection (see Khanna and Liu, 2006). Most of the approaches apply Hidden Markov Models.

3. Method materialization and access prediction - basic concepts

The HMM recomputation technique, proposed in this paper, is general, but we implemented it for the so called *hierarchical materialization* (Bębel and Wrembel, 2001, Jezierski et al., 2003, and Jezierski, Masewicz and Wrembel, 2004). The HMM recomputation further extends our previous work, where we proposed the PMAP recomputation technique (Masewicz et al., 2006). The essential ideas behind the hierarchical materialization, the PMAP recomputation, and the Hidden Markov Model are presented in this section.

3.1. Hierarchical materialization

In the *hierarchical materialization* (similarly as in a traditional approach), the first execution of m_i, for object o_i and with a given set of input argument values, materializes the result of m_i. Additionally, the hierarchical materialization entails materialization of intermediate results of other methods transitively called from m_i. The intermediate materialized results are used for recomputing m_i, after the update of the m_i base object. In this way, the recomputation time of m_i is reduced. The idea behind hierarchical materialization is illustrated with the example below.

Example 1 *Let us consider a simplified CAD design of a personal digital assistant (PDA). This design is represented by object* $m515$ *(the instance of the* PDA

*class), which is composed of objects mb*100*, sc*100*, and dsp*100 *(the instances of the MainBoard, SoundCard, and Display classes, respectively); mb*100 *is further composed of cpu*33*, etc.; each of these classes has method power*() *that computes and returns power consumption of a certain object; a collaboration diagram (in the UML notation) between the instances of the PDA classes is shown in Fig. 1; the value of power*() *for object m*515 *is computed as the sum of power consumed by its components; the value of power*() *for mb*100 *is computed as the sum of power consumed by the object itself and its component cpu*33*, etc.*

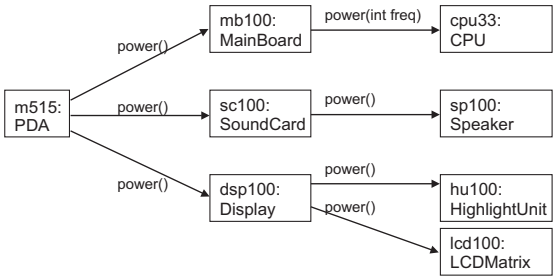

Figure 1. An example of a collaboration diagram of a PDA

Let us further assume that the power() *method was invoked for object m*515 *and it was materialized hierarchically. In our example, the hierarchical materialization results in materializing also mb*100.*power*()*, sc*100.*power*()*, and dsp*100.*power*()*.*

*Having materialized the methods, let us assume that the component object cpu*33 *was replaced with another central processing unit using 133MHz clock, instead of 33MHz. This change results in higher power consumption of main board mb*100 *and of the whole PDA m*515*. As a consequence, the materialized values of m*515.*power*() *and mb*100.*power*() *have to be invalidated and recomputed during next invocation. However, during the recomputation of m*515.*power*()*, the unaffected materialized results of sc*100.*power*() *and dsp*100.*power*() *can be used.*

Hierarchical materialization can have various applications, e.g., (1) in object distributed environments (e.g. Corba) for synchronizing replica objects; (2) in multimedia databases and multimedia data warehouses (see Messaoud, Boussaid and Rabaséda, 2004) for computing parameters of images; (3) in object-relational data warehouses (see Huynh, Mangisengi and Tjoa, 2000) as a technique for materializing object views.

3.2. PMAP

The PMAP recomputation technique is used for predicting forthcoming operations on materialized methods and their base objects. To this end, for every

materialized method m_i the system logs the history of m_i reads (further noted as `R`) and its base object o_i updates (further noted as `U`). Operations on m_i and o_i are represented as the so called workloads. A *workload* is the set of `R` and `U` operations executed within a given time period (e.g. a day, an hour). The number of `R` and `U` operations in a given workload is called *workload length*. Workloads are ordered by time. The interleaved sequences of `R` and `U` within a workload form the so called *workload pattern*. Because workload pattern may change in time, multiple workloads having similar patterns are combined into the so called *workload sets*, see Masewicz et al. (2006). The number of operations in a workload set is further called *workload set length* and is denoted ϑ.

Predicting forthcoming operations on materialized methods and their base objects is based on workload set. It works as follows. Firstly, the longest `[UU...U]` sequence in the workload set is found. The *sequence length*, denoted λ, is defined as the number of consecutive `U` operations.

Secondly, the system computes the frequency of a sequence of length l in the workload set. This frequency is computed for sequences of length 1 to λ. Let N_U^i be the number of `[U...]` sequences of length i, where $i = \{1, 2, \ldots, \lambda\}$. The *frequency* ρ_U^i of `[U...]` sequences of length i in a workload set is expressed by (1). In this formula, $\sum_{n=i}^{\ell} N_U^n$ represents the number of `U` sequences of length greater than or equal to i; $\sum_{n=1}^{\ell} N_U^n$ represents the total number of `U` sequences:

$$\rho_u^i = \frac{\sum_{n=i}^{\lambda} N_U^n}{\vartheta + \sum_{n=1}^{\lambda} N_U^n} \,. \tag{1}$$

Next, the system checks if the frequency is greater than a given (parameterized) value. If so, m_i is left as invalidated since it is likely that the next operation in the current workload is `U`. Otherwise m_i is recomputed immediately since it is likely that the next operation in the current workload is `R`. The presented concept is illustrated with the example below.

EXAMPLE 2 *Let us consider past daily workloads from four days, as shown below:*

```
[URUUURRRUR  UUUUURUUUR  UURRRRURUU  URURRUUURU]
 <- day1 ->  <- day2 ->  <- day3 ->  <- day4 ->
```

For each of these four daily workloads independently we compute coefficients representing shares of `U` *operations. In our example, the coefficients are as follows:* 5/10 *(day*1*),* 8/10 *(day*2*),* 5/10 *(day*3*), and* 6/10 *(day*4*). Let us assume that all consecutive workloads whose coefficients differ by at most 10% are included into the same workload set. In our example, this condition is fulfilled by workloads from day*3 *and day*4 *that are included in workload set denoted as W. The other two workloads are left separated. Further analysis is performed for W whose workload set length* $\vartheta = 20$*. In W, we search for the longest* `U` *sequence. In our example it is* `[UUU]`*, i.e.* $\lambda = 3$*. Next, we count the number*

of occurences of sequence `[UUU]` *in W. Such a sequence appears twice, which we note as* $N_U^3 = 2$. *Next, we count the number of sequences shorter by 1, i.e.* `[UU]`, *i.e.* $N_U^2 = 1$. *Finally, we compute the number of* `[U]` *sequences, i.e.* $N_U^1 = 3$.

For every detected sequence of `[U...]` *we compute its frequency* ρ_U^i, *according to Formula 1. Thus,* $\rho_U^3 = 1/13$, $\rho_U^2 = 3/26$, *and* $\rho_U^1 = 3/13$.

Let us now assume that a new workload arrives with the first operation `U`. *It invalidates the value of* m_i. *Now the system must decide whether the next operation in the new workload is* `[U]` *(no need to recompute* m_i*) or* `[R]` *(*m_i *has to be recomputed). Based on the past workload the system computes the probable frequency of the next* `U` *operation. In our example the frequency* $\rho_U^2 = 3/26$ *and if it is greater than the parameterized value* ρ, *then* m_i *is recomputed immediately. Otherwise* m_i *is left invalid.*

3.3. Hidden Markov Model

A Hidden Markov Model (HMM) (see Rabiner, 1989) is an automaton composed of a finite set of states. With each of these states, a probability of transition to all other states is defined. When the automaton enters one of its states, it emits a symbol according to some probability distribution (defined separately for each of the states). Next, with an appropriate transition probability, it changes the state and the process repeats.

EXAMPLE 3 *As a simple example, let us consider an automaton with two states* S_1 *and* S_2, *as shown in Fig. 2. From state* S_1 *one can reach state* S_2 *with probability equal 0.2 and state* S_1 *with probability equal 0.8. Being in* S_1 *the automaton can emit either symbol* `U` *or* `R` *with probability equal 0.4 or 0.6, respectively.*

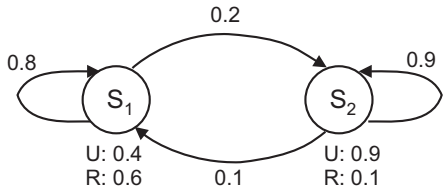

Figure 2. An example of automaton with transition and emission probabilities

From S_2 *the automaton can reach state* S_1 *with probability equal 0.1 and* S_2 *with probability equal 0.9. Being in* S_2 *the automaton can emit either symbol* `U` *or* `R` *with probability equal 0.9 or 0.1, respectively.*

There are two important assumptions for the HMM. Firstly, the transition probabilities do not depend on the history of state transitions i.e., they have the so-called Markov property. Secondly, an external observer sees only the sequence of generated symbols, but he/she does not see the sequence of states

which generated these symbols. Several algorithms were developed for the HMM (see Rabiner, 1989), including:

- the Forward-Backward algorithms that may be applied to calculating the probabilities of obtaining a given sequence of symbols;
- the Viterbi algorithm that may be applied to calculating the most likely sequence of hidden states that generated a given sequence of symbols;
- the Baum-Welch algorithm that may be applied to calculating the transition and emission probabilities based on the sequence of observed emitted symbols.

The three algorithms assume that the number of states is known.

Possible applications of the HMM include: speech recognition, image recognition, DNA sequence analysis, automatic translation of texts, weather forecasting.

4. HMM recomputation technique

We will now describe our approach to selecting a method recomputation technique that is most suitable for a current workload. This technique is self adaptable to a workload that may dynamically change. As mentioned in Section 1, the technique, called the HMM recomputation is based on the Hidden Markov Model, used for describing the behavior of a system. The system is composed of a set of applications or user sessions, each of which can either read a given method value or update a base object. Thus, an application can be represented by a state that can emit with a given probability either symbol `R` or `U`. The order, in which applications are executed, is represented by transition probabilities between states.

A simple mapping of a system to the HMM is shown in Fig. 3. This system is composed of n applications, represented by states $S_0, S_1, \ldots, S_n$. Application S_0 reads a given method value with probability equal 0.3 and updates a base object with probability equal 0.7. The probability that S_1 will be executed next equals 0.7 and the probability that S_0 will be executed again equals to 0.3, etc.

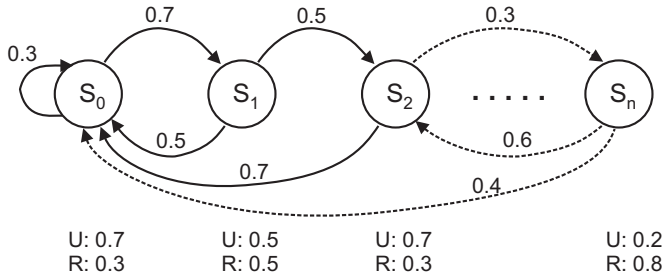

Figure 3. An example HMM automaton representing a simple system composed of n applications executing either `R` or `U` operation

In order to predict an operation that will be executed in the system as the next one, we apply the Baum-Welch and Forward-Backward algorithms. First, the HMM is trained, based on the history of operations, using the Baum-Welch algorithm. The obtained model is then subsequently used for calculating the probabilities of forthcoming operations using the Forward-Backward algorithms. If the calculated probability of an operation is greater than 0.5, then it is assumed that this operation will be executed as the next one.

In order to keep the HMM up-to-date we apply a technique similar to the one used in the PMAP, i.e. the model is rebuilt each time we observe a significant change in the characteristics of the observed sequence of operations. In order to detect such changes, we divide the observed sequence of operations into periods of a fixed length l. At the time the model is built, we also estimate the probabilities of occurrences of both types of operations (`R` and `U`), based on the frequency of their occurrence and store them in a database. When the subsequent periods finish, we estimate these probabilities once again, based on the sequence of operations from the last period. If the change of probabilities is greater than the specified threshold t, the HMM is rebuilt. In order to illustrate this technique let us consider the following example.

EXAMPLE 4 *Assume that* $l = 5$ *and* $t = 0.1$*. Moreover, assume, that the observed sequence of operations from the last period before the model has been built is* $H_1 = RRURR$*. Also, let us assume, that the sequence of operations from the last period is* $H_2 = URRUU$*. The probabilities estimated, based on the sequence* H_1 *are equal to* $P(U) = 0.2$ *and* $P(R) = 0.8$*, whereas for the sequence* H_2 *the estimated probabilities are equal to* $P(U) = 0.6$ *and* $P(R) = 0.4$*. Because the difference in probabilities is greater than the predefined threshold* $t = 0.1$*, the model needs to be rebuilt.*

The presented technique for predicting the forthcoming operations is executed every time when the last observed operation is `U` (recall that update operations invalidate the results of materialized methods). If the predicted operation is also `U`, then the rematerialization of the invalidated method m_i is not executed, because the recomputed result would be invalidated again by the forthcoming `U` operation. If the predicted operation is `R`, then the rematerialization of m_i is executed before providing its value.

Let us notice that the PMAP, presented in Section 3.2, is a special case of the solution based on the HMM. In the PMAP, prediction is based on operation frequencies. These frequencies are calculated based on the history of operations performed on a given object. The PMAP can be represented by the HMM that contains states corresponding to the series of updates of a given length. We will now present a method for constructing the HMM representing the PMAP. We will denote such a HMM as $PMAP_{HMM}$.

$PMAP_{HMM}$ is composed of the following states:

- A starting state, denoted as *Start* that emits only one, meaningless, symbol `B` (probability of emitting `B` is always 1). Symbol `B` does not influence

predictions, since it appears only once at the beginning of a historical workload. The HMM never returns to the state *Start*. In order to perform calculations based on the HMM representation of PMAP, symbol `B` must be appended at the start of the observed sequence of symbols.

- State $S_n - U$ that represents the following event: an update operation has just been emitted (emission probability of `U` is 1) and previous n updates have also been emitted.
- State $S_n - R$ that represents the following event: a read operation has just been emitted (emission probability of `R` is 1) and previous n updates have also been emitted.

Transition probabilities are calculated using the following rules:

- $P(start \rightarrow S_0 - U)$ - probability of emitting an update, if no updates have been observed;
- $P(start \rightarrow S_0 - R)$ - probability of emitting a read operation, if no updates have been observed;
- $P(S_n - U \rightarrow S_{n+1} - U)$ - the probability of emitting `U`, if $n+1$ updates have been observed;
- $P(S_n - U \rightarrow S_{n+1} - R)$ - the probability of emitting `R`, if $n+1$ updates have been observed;
- $P(S_n - R \rightarrow S_0 - U)$ - the probability of emitting `U`, if no updates have been observed;
- $P(S_n - R \rightarrow S_0 - R)$ - the probability of emitting `R`, if no updates have been observed;
- $P(S_{max} - U \rightarrow S_0 - U)$ (max denotes the longest, predefined sequence of `U`) - the probability of emitting `U`, if no updates have been observed;
- $P(S_{max} - U \rightarrow S_0 - R)$ - the probability of emitting `R`, if no updates have been observed;
- probability of all other transitions is equal to 0.

In order to illustrate the HMM representation of the PMAP let us consider the following example.

Example 5 *Fig. 4 presents an example of HMM representing the PMAP, assuming that the maximum length of the sequence of updates equals 4. Thus, we have four states $S_i - U$ and four states $S_i - R$ ($i = 0, \ldots, 3$). The figure presents all of the states as well as examples of their transition and emission probabilities.*

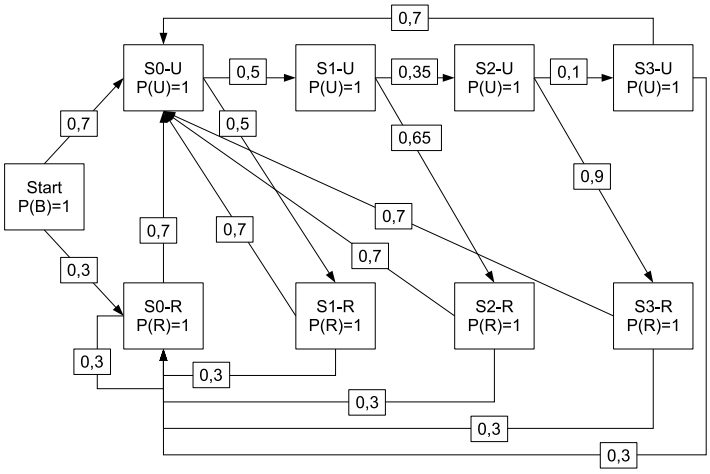

Figure 4. An example of HMM representation of the PMAP for the length of an update sequence equal 4

The PMAP has a low computation complexity and it offers good prediction accuracy (see Masewicz et al., 2006). For these reasons, further in this paper, we relate the characteristics of the HMM recomputation to the PMAP recomputation.

5. Experimental evaluation

The proposed HMM recomputation technique was evaluated by experiments. Its performance characteristics were compared to four other recomputation techniques discussed in this paper, namely to: the deferred, immediate, PMAP, and random recomputations. In the random recomputation an invalid materialized result of method m_i is recomputed randomly.

All the five recomputation techniques were implemented in Java in an object database FastObjects t7 9.0 (see Versant, 2006). FastObjects was used also as a storage system for test data, whose size reached 2GB. The experiments were run on a PC with AMD Athlon 2GHz, with 512 MB of RAM, under WindowsXP. Execution times of original non-materialized methods were equal 1 second.

For the purpose of evaluating the aforementioned recomputation techniques we used two efficiency measures, namely: (1) system time overhead (STO) and (2) user time overhead (UTO). The STO represents an overall time spent by the system for: (1) predicting the forthcoming operation, (2) executing a method (if needed), and (3) accessing a materialized value (if available). The UTO represents an overall time that a user's application (session) has to wait for obtaining the result of a called method. Moreover, in order to compare the quality of prediction we measured the number of hits. A hit is a correctly predicted operation.

The experiments were conducted for workloads of different lengths (ϑ) ranging from 100 to 5000 interleaved `R` and `U` operations. The number of base object updates in every workload was parameterized. We tested the performance for five scenarios with 10%, 25%, 50%, 75%, and 90% of update operations in a workload. In every test a workload pattern was randomly generated. In this paper we present average time characteristics for 25% of update operations (75% of read operations) in a workload. For other numbers of updates, we obtained characteristics that are similar to the characteristics presented in this paper. The number of applications running in parallel was parameterized and ranged from 1 to 10.

Every experiment was run 10 times for the purpose of testing its repeatability. The charts presented in this paper show average times of 10 runs of a given experiment. In all of the experiments the standard deviation oscillated between 1% and 2% of an average value.

5.1. Experiment 1

This experiment measured the STO and the UTO of the five recomputation techniques mentioned above and it was run for the following settings: (1) the state graph, i.e. transitions between applications (used by the HMM recomputation), was fixed and known; (2) the probabilities of generating `R` and `U` operations (used by the HMM recomputation) were fixed and known for all applications.

The results for five applications running in parallel are shown in Figs. 5a and 5b that present the STO and UTO, respectively. As we can observe from Fig. 5b, the immediate recomputation technique (denoted as *immediate*) introduces the lowest UTO, which is quite obvious since a method recomputation immediately follows a base object update. Thus, a method value is ready for an application before the latter requests the value. The deferred recomputation technique (denoted as *deferred*) introduces the highest UTO.

From the system point of view, the immediate recomputation is the most costly (see the *immediate* curve in Fig. 5a). It is because in many cases a previously recomputed result is invalidated by a forthcoming `U` operation. Moreover, the deferred recomputation introduces the lowest STO (see curve *deferred* in Fig. 5a since the system executes and rematerializes a method only when its value is requested.

Comparing the PMAP and HMM recomputation techniques, we can observe that the HMM recomputation offers lower UTO than the PMAP (see curves *HMM* and *PMAP* in Fig. 5b). It is caused by a better prediction accuracy of the HMM recomputation that predicts the right operation more frequently than PMAP. However, by analyzing the chart from Fig. 5a we can observe that the STO is higher for the HMM recomputation than for the PMAP.

The random recomputation technique (denoted as *random*) yields substantially worse UTO and STO than the HMM and PMAP. This fact proves that

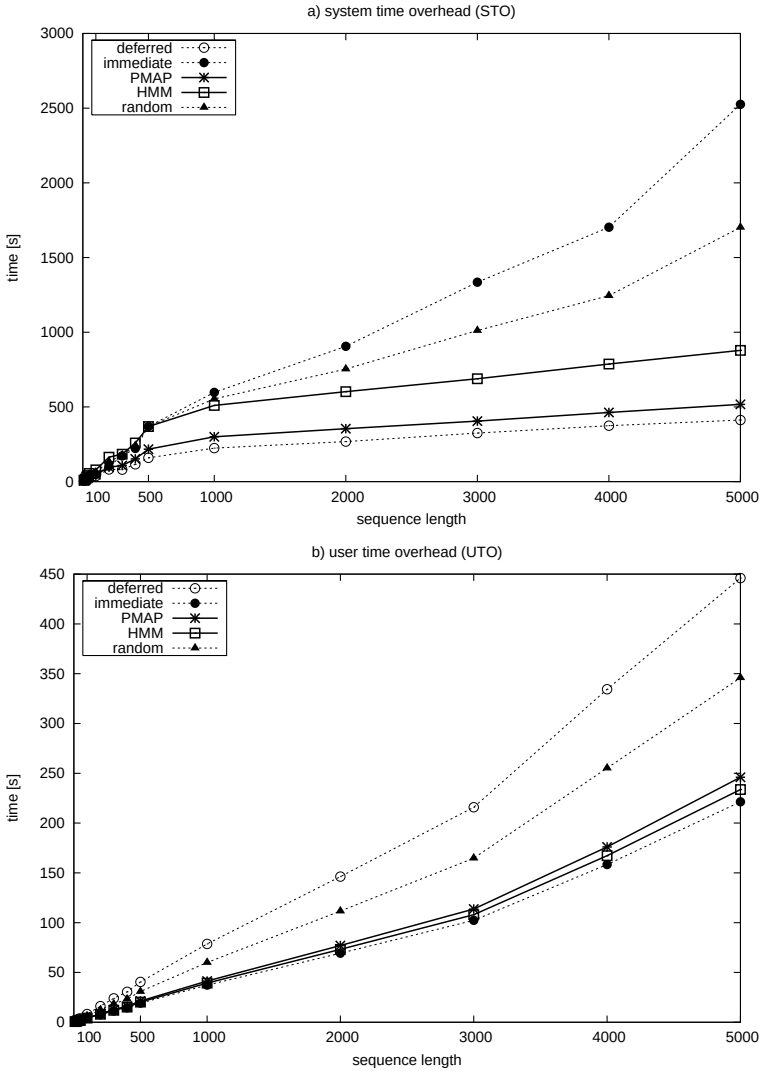

Figure 5. The STO and UTO (five parallel applications; 25% of `U` operations in workloads; workload length from 100 to 5000; the state graph and the probabilities of generating operations are fixed and known)

both the PMAP and HMM recomputation techniques offer a reasonable improvement in an overall performance of a system with materialized methods.

5.2. Experiment 2

In this experiment the state graph (i.e. the transitions between applications) was unknown, therefore it was built by the Baum-Welsh and Viterbi algorithms based on the observed `R` and `U` operations generated by applications. The probabilities of generating `R` and `U` operations were fixed and known for all applications.

The results for five applications running in parallel are shown in Figs. 6a and 6b, presenting the STO and UTO, respectively. The charts compare the performance of the HMM recomputation with state graph building (denoted as *HMM+gb*) to the performance of the recomputation techniques from Experiment 1.

As we can observe from Fig. 6a, the HMM recomputation with graph building is more computationally costly with respect to STO as compared to the HMM recomputation. We can also notice that after applying the HMM recomputation with graph building, the UTO remained the same as in Experiment 1 (see Fig. 5b). It means that the HMM with graph building well predicts the transitions between states (applications).

5.3. Experiment 3

This experiment measured the STO, UTO, and prediction accuracy. It was run for the following settings: (1) the state graph was fixed and known; (2) the probabilities of generating `R` and `U` operations were fixed and known for five applications; (3) the probabilities of generating `R` and `U` operations were random for the other five applications.

The STO and UTO are shown in Figs. 7a and 7b, respectively. From the charts we can clearly observe that the STO of the PMAP recomputation is much lower than the STO of the HMM recomputation, similarly as in Experiment 1. Moreover, the UTO of the PMAP is also much lower than the UTO of the HMM, which is caused by the bigger number of hits, returned by the PMAP. Similarly as in Experiment 1, the immediate and random recomputations offer the highest STO. When analyzing the UTO for the random recomputation we can notice that it is slightly lower than for the HMM recomputation. This is caused by worse prediction accuracy of the HMM recomputation for five random applications.

Fig. 8 presents the prediction accuracy of the HMM, PMAP, and random recomputation with respect to the number of applications, whose behavior was random. The prediction accuracy is represented by the percentage of hits. The number of applications randomly generating `R` and `U` operations ranged from 0

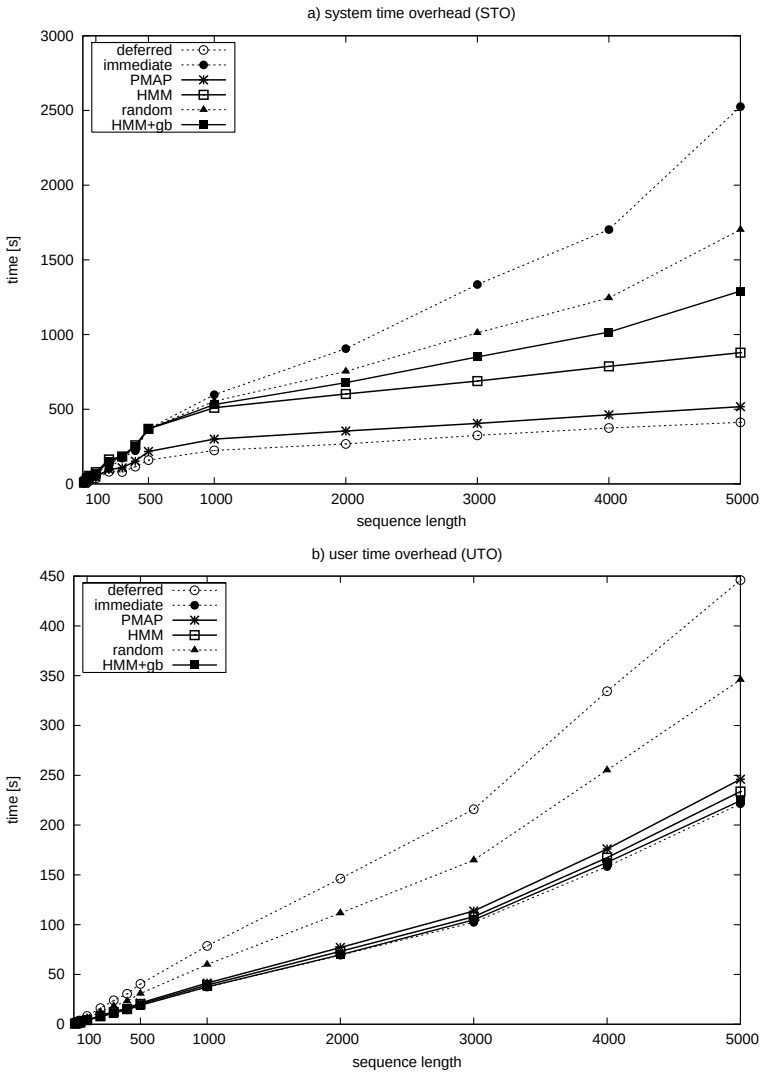

Figure 6. The HMM with state graph building: the STO and UTO (five parallel applications; 25% of U operations in workloads; workload length from 100 to 5000; the state graph is unknown; the probabilities of generating operations are fixed and known)

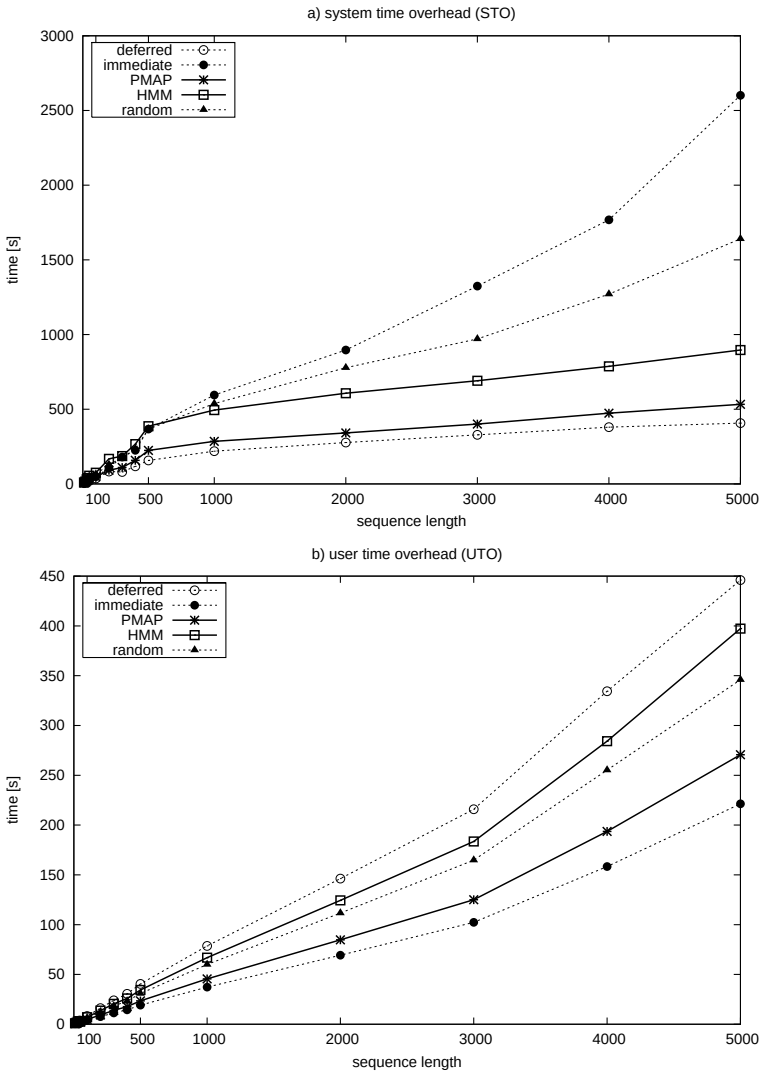

Figure 7. The STO and UTO (ten parallel applications; 25% of U operations in workloads; workload length from 100 to 5000; the state graph is fixed and known; the probabilities of generating `R` and `U` operations are random for five applications)

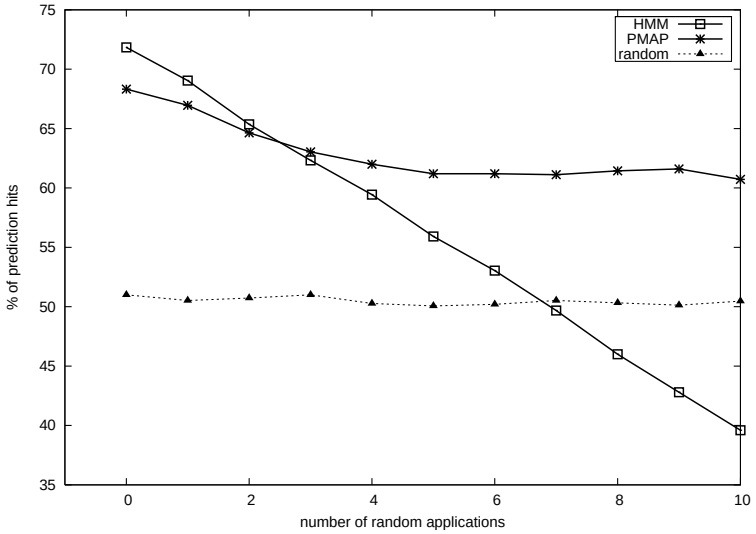

Figure 8. Prediction accuracy for the HMM, PMAP, and random recomputations (ten parallel applications; 25% of U operations in workloads; workload lenght 5000; the state graph is fixed and known; the number of random applications ranges from 0 to 10)

(i.e. probabilities of generating `R` and `U` were fixed and known for all applications) to 10 (i.e. probabilities of generating `R` and `U` were unknown for all applications).

From Fig. 8 we can observe that for applications, whose behavior was known (0 of random applications), the HMM recomputation offers the highest hit number, i.e. approximately 72%, whereas the PMAP offers approximately 68%, and the random recomputation offers approximately 50%. With an increase in the number of random applications the hit number by the HMM substantially (almost linearly) decreases down to approximately 40% for 10 random applications. The hit number by the PMAP decreases slightly, down to approximately 60% for 10 random applications. For four and more random applications, the number of hits by the HMM becomes lower than by the PMAP. Such behaviour can be explained by the fact that the PMAP analyzes the history of workloads, whereas the HMM does not.

5.4. Experiment 4

In this experiment the state graph (i.e. the transitions between applications) was unknown, and so it was built by the Baum-Welsh and Viterbi algorithms. Moreover, the probabilities of generating `R` and `U` operations were unknown for a parameterized number of applications that ranged from 0 to 10. This experiment measured the STO, UTO, and prediction accuracy.

Figs. 9a and 9b show the STO and UTO, respectively. These charts describe a scenario where the probabilities of generating `R` and `U` operations were fixed and known for five applications and for the other five applications they were random.

As we can observe from Fig. 9a, the HMM recomputation with graph building (see the curve *HMM+gb*) is more computationally costly with respect to the STO when compared to the HMM and PMAP recomputations. The UTO for the HMM with graph building is lower than for the HMM. This is caused by a better prediction accuracy of the HMM with graph building than of the HMM, i.e. the number of hits by the HMM with graph building is greater than by the HMM. However, applying the HMM with graph building results in higher UTO when compared to the HMM and PMAP recomputations.

Fig. 10 presents the prediction accuracy of the PMAP, HMM, and random recomputations with respect to the number of applications, whose behavior was random. The number of applications randomly generating `R` and `U` operations ranged from 0 to 10.

From Fig. 10 we can observe that the HMM with graph building offers a prediction accuracy oscillating around approximately 70%, regardless of the number of random applications. With this regard, the HMM with graph building is much better than the other recomputation techniques. Such a good prediction accuracy was achieved since the prediction model was rebuilt as the result of a workload change. The much worse prediction accuracy of the HMM was caused by the fact that the prediction model built once remained unchanged even if a workload changed. As a consequence, the prediction model of the HMM recomputation did not describe correctly the current workload.

5.5. Experiment summary

By analyzing the results of the above four experiments we can draw the following conclusions:

- the computation complexity (i.e. the STO) is greater for the HMM and HMM with graph building recomputations than for the PMAP;
- the HMM recomputation guarantees **lower** UTO than the PMAP when the probabilities of operations requested by applications are **known**, regardless of the fact whether the state graph is known or not (see Experiments 1 and 2);
- the HMM and the HMM with graph building recomputations guarantee **higher** UTO than the PMAP when the probabilities of operations requested by applications are **unknown**, regardless of the fact whether the state graph is known or not (see Experiments 3 and 4);
- the HMM with graph building recomputation assures the best prediction accuracy of all the tested recomputation techniques when at least one random application is running in the system (see Experiment 4);

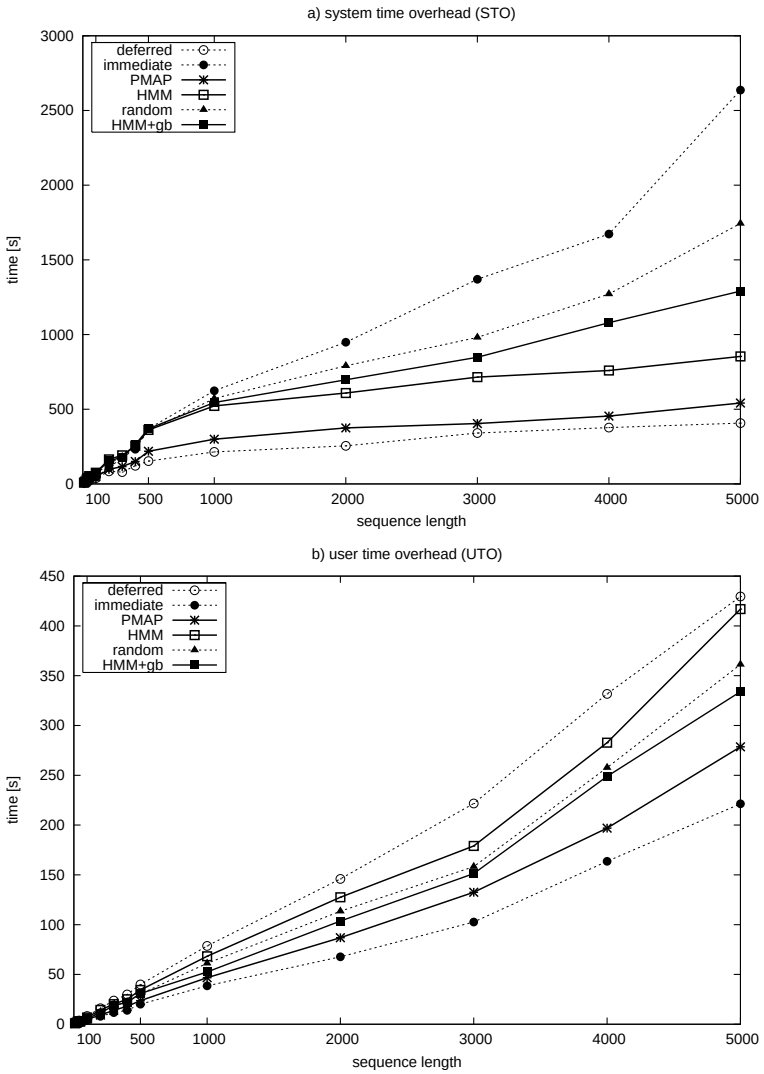

Figure 9. The HMM with state graph building: the STO and UTO (ten parallel applications; 25% of U operations in workloads; workload length from 100 to 5000; the state graph is unknown; the probabilities of generating operations are fixed and known for five applications)

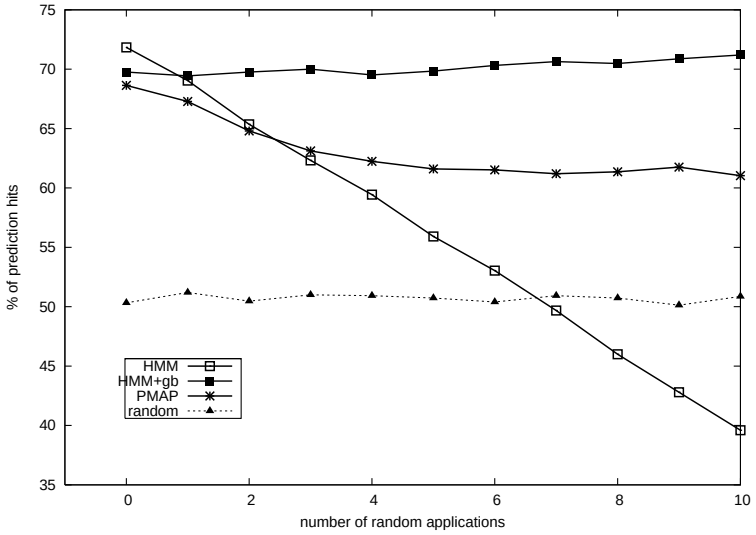

Figure 10. Prediction accuracy for the HMM with graph building, HMM, PMAP, and random recomputations (ten parallel applications; 25% of U operations in workloads; workload length 5000; the state graph is unknown; the number of random applications ranges from 0 to 10)

- the HMM recomputation assures the best prediction accuracy when no random applications are running in the system (see Experiment 4);
- the HMM, HMM with graph building, and PMAP recomputations assure substantially better performance with respect to UTO and STO than random recomputation. This fact well justifies the application of these recomputation techniques.

6. Summary

In this paper we presented a technique, called HMM recomputation, for recomputing materialized method results. The main goal of this technique is to choose the most suitable moment of recomputing a materialized method result for a given workload, so that an overall system performance should increase. To this end, the proposed HMM recomputation predicts forthcoming operations on methods and their base objects. The prediction model is based on the Hidden Markov Model.

The proposed technique has been applied to an object system. However, it is a general technique and it can also be applied in other systems, e.g. in distributed databases and distributed computing environments for synchronizing replicas, or in data warehouses for refreshing materialized views.

The proposed technique was implemented and experimentally evaluated. As a reference we chose four other possible recomputation techniques, namely: the immediate, deferred, random, and PMAP, which is also based on a prediction model. The experiments show that: (1) after applying the HMM, HMM with graph building as well as PMAP recomputations the overall system performance increases; (2) the prediction accuracy depends on the number of random applications running in a system, however the HMM with graph building offers the most promising characteristics.

For the work presented in this paper some parameters of the HMM and HMM with graph building recomputation techniques were arbitrarily fixed. Such parameters include: (1) the length of historical workloads, (2) the probability threshold, and (3) workload change threshold. Finding appropriate values of these parameters remains an open problem and task for future development.

The length of historical workloads being analyzed has influences prediction accuracy and time of computing the prediction. If the workload is too long, then the HMM recomputation takes too long. On the contrary, if the workload is short, then prediction accuracy decreases. Therefore, there should be a balance between prediction accuracy and computation time that has to be found.

Moreover, the HMM recomputation recomputes an invalid method when the probability of a forthcoming operation is greater than a given threshold. This threshold influences system performance. If it is too high, then applications will wait for method values more frequently, but the system will be less charged with method recomputations. On the contrary, if the threshold is too low, then a system will be more charged with recomputations, but applications will wait shorter for method values. Therefore, finding the right threshold is another important issue.

In a real system applications may dynamically change their behavior, i.e. probabilities of executed operations on methods and objects. As a consequence, an HMM model built once may no longer describe a current behavior of applications. For this reason another important issue is to find the right threshold in application behaviors, triggering the rebuilding of an HMM model.

Yet another open problem is to consider priorities for applications, so that applications requesting at the same moment contradictory recomputation techniques can be scheduled according to their priorities.

Last but not least, an important research problem is to develop a cost model that will allow for developing a kind of self tuning system, able to automatically adjust the three aforementioned parameters according to the current behavior of applications.

References

Ali, M.A., Fernandes, A.A.A. and Paton, N. (2000) Incremental maintenance of materialized oql views. *Proceedings of ACM Int. Workshop on Data Warehousing and OLAP (DOLAP)*. ACM Press, 41–48.

Arigon, A.M., Tchounikine, A. and Maryvonne, M. (2006) Handling multiple points of view in a multimedia data warehouse. *ACM Transactions on Multimedia Computing, Communications and Applications* **2** (3), 199–218.

Bédard, Y., Rivest, S. and Proulx, M. J. (2007) Spatial on-line analytical processing (solap): Concepts, architectures and solutions from a geomatics engineering perspective. In: R. Wrembel and C. Koncilia, eds., *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, 298–319. Idea Group Inc.

Bertino, E. (1991) Method precomputation in object-oriented databases. *SIGOS Bulletin* **12** (2,3), 199–212.

Bębel, B. and Wrembel, R. (2001) Hierarchical materialisation of methods in oo views: Design, maintenance, and experimental evaluation. *Proceedings of ACM Int. Workshop on Data Warehousing and OLAP (DOLAP)*. ACM Press, 77–84.

Bukhres, O.A. and Elmagarmid, A. (1995) *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications*. Prentice Hall.

Cattell, R.G.G., Barry, D., Berler, M., Eastman, J., Jordan, D., Russel, C., Shadow, O., Stanienda, T., and Velez, F. (2000) *Object Database Standard: ODMG 3.0*. Morgan Kaufmann Publishers.

Czejdo, B., Eder, J., Morzy, T. and Wrembel, R. (2001a) Design of a data warehouse over object-oriented and dynamically evolving data sources. *DEXA Workshop on Parallel and Distributed Databases*. IEEE Computer Society, 128–132.

Czejdo, B., Eder, J., Morzy, T. and Wrembel, R. (2001b) Designing and implementing an object-relational data warehousing system. *IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems (DAIS)*. Kluwer, 311–316.

Deng, X. and Ali, H. (2004) A hidden Markov model for gene function prediction from sequential expression data. *Proceedings of IEEE Computational Systems Bioinformatics Conference (CSB)*. IEEE Computer Society, 670–671.

Dongshan, X. and Junyi, S. (2002) A new Markov model for web access prediction. *Computing in Science and Engineering* **4** (6), 34–39.

Drton, M., Marzban, C., Guttorp, P. and Schaefer, J.T. (2003) A Markov chain model of tornadic activity. *AMS Journals Online. Monthly Weather Review* **131** (12), 2941–2953.

Eder, J., Frank, H. and Liebhart, W. (1994) Optimization of object-oriented queries by inverse methods. *Int. East/West Database Workshop. Workshops in Computing*. Springer Verlag, 109–121.

Fankhauser, P., Gardarin, G., Lopez, M., Munoz, J. and Tomasic, A. (1998) Experiences in federated databases: From iro-db to miro-web. *Proceedings of Int. Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann, 655–658.

GARDARIN, G., SHA, F. and TANG, Z.H. (1996) Calibrating the query optimizer cost model of iro-db, an object-oriented federated database system. *Proceedings of Int. Conference on Very Large Data Bases (VLDB)*. Morgan Kaufman, 378–389.

GOPALKRISHNAN, V., LI,Q. and KARLAPALEM, K. (2000) Efficient query processing with associated horizontal class partitioning in an object relational data warehousing environment. *Proceedings of Int. Workshop on Design and Management of Data Warehouses (DMDW)*. CEUR-WS.org.

GORAWSKI, M., and KAMIŃSKI, M. (2006) On-line balancing of horizontally-range-partitioned data in distributed spatial telemetric data warehouse. *DEXA Workshops*. IEEE Computer Society, 273–280.

GORAWSKI, M. and MALCZOK, R. (2005) Updating aggregation tree in distributed spatial telemetric data warehouse. *Euromicro Workshop on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE Computer Society, 329–336.

HUYNH, T.N., MANGISENGI, O. and TJOA, A.M. (2000) Metadata for object-relational data warehouse. *Proceedings of Int. Workshop on Design and Management of Data Warehouses (DMDW)*. CEUR-WS.org.

JEZIERSKI, J., MASEWICZ, M. and WREMBEL, R. (2004) Prototype system for method materialisation and maintenance in object-oriented databases. *Proc. of ACM Symposium on Applied Computing (SAC)*. ACM Press, 1323–1327.

JEZIERSKI, J., MASEWICZ, M., WREMBEL, R. and CZEJDO, B. (2003) Designing storage structures for management of materialised methods in object-oriented databases. *Int. Conference on Object-Oriented Information Systems (OOIS)*. **LNCS 2817**, Springer Verlag, 202–213,

JHINGRAN, A. (1991) Precomputation in a complex object environment. *Proceedings of Int. Conference on Data Engineering (ICDE)*. IEEE Computer Society, 652–659.

KANDASWAMY, V. (2006) Object relational data warehousing: Viewing linked data. Retrieved September 10, 2006, from *http://www.datawarehouse.com/article/?articleid=3134*.

KAXIRAS, S. and YOUNG, C. (2000) Coherence communication prediction in shared-memory multiprocessors. *Int. Symposium on High-Performance Computer Architecture (HPCA)*. IEEE Computer Society, 156–167.

KEMPER, A., KILGER, C. and MOERKOTTE, G. (1994) Function materialization in object bases: Design, realization, and evaluation. *KDE* **6** (4), 587–608.

KEMPER, A. and MOERKOTTE, G. (1994) *Object-Oriented Database Management: Applications in Engineering and Computer Science.* Prentice Hall.

KHANNA, R. and LIU, H. (2006) System approach to intrusion detection using hidden Markov model. *Proceedings of Int. Conference on Wireless Communications and Mobile Computing*. ACM Press, 349–354.

Kim, H.H. and Park, S.S. (2003) Building a web-enabled multimedia data warehouse. *Web and Communication Technologies and Internet-Related Social Issues (HSI)*. **LNCS 2713**, Springer Verlag, 594–600.

Konovalov, A. (2002) Object-oriented data model for data warehouse. *Proc. of East European Conference Advances in Databases and Information Systems (ADBIS)*. **LNCS 2435**, Springer Verlag, 319–325.

Kratky, M., Stolfa, S., Snasel, V. and Vondrak, I. (2005) Efficient searching in large inheritance hierarchies. *Proceedings of Int. Conference on Database and Expert Systems Applications (DEXA)*. **LNCS 3588**, Springer Verlag, 940–952.

Kuno, H.A. and Rundensteiner, E.A. (1998) Incremental maintenance of materialized object-oriented views in multiview: Strategies and performance evaluation. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* **10** (5), 768–792.

Liu, Y. and Teitelbaum, T. (1995) Caching intermediate results for program improvement. *ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*. ACM Press, 190–201.

Liu, Y.A., Stoller, S.D. and Teitelbaum, T. (1998) Static caching for incremental computation. *ACM Transactions on Programing Languages and Systems* **20** (3), 546–585.

Loomis, M.E.S. (1995) *Object Database the Essentials*. Addison-Wesley.

Loomis, M.E.S. and Chaudhri, A.B. (1998) *Object Databases in Practice*. Prentice Hall.

Masewicz, M., Wrembel, R., Stabno, M. and Staniszewski, R. (2006) Pmap: Framework to predicting method access patterns for materialized methods. *Proceedings of Int. Conference on Advances in Information Systems (ADVIS)*. **LNCS 4243**, Springer Verlag, 14–323.

Messaoud, B.R., Boussaid, O. and Rabaséda, S. (2004) A new olap aggregation based on the ahc technique. *Proceedings of ACM Int. Workshop on Data Warehousing and OLAP (DOLAP)*. ACM Press, 65–72.

OMG (2006) Corba. Retrieved November 10, 2006 from *http://www.omg.org/technology/documents/formal/corba_2.htm*.

Oral, M. and Kettani, O. (1989) A mathematical programming model for market share prediction. *International Journal of Forecasting* **5** (1), 59–68.

Pâris, J.F., Amer, A. and Long, D.D.E. (2003) A stochastic approach to file access prediction. *Proceedings of the Int. Workshop on Storage Network Architecture and Parallel I/Os*. ACM Press, 36–40.

Pugh, W. and Teitelbaum, T. (1989) Incremental computation via function caching. *ACM Principles of Programming Languages (POPL)*. ACM Press, 315–328.

Rabiner, L.R. (1989) A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. of the IEEE* **77** (2), 257-286.

VERSANT (2006) Fast()objects t7. Retrieved May 30, 2006, from *http://www.versant.com/developer/resources/fastobjects/overview*.

YU, S., ATLURI, V. and ADAM, N. (2006) Preview: Optimizing view materialization cost in spatial data warehouses. *Proceedings of Int. Conference on Data Warehousing and Knowledge Discovery (DaWaK)*. **LNCS 4081**, Springer Verlag, 45–64,