A PREDICTIVE AUTOMATIC TUNING SERVICE FOR OBJECT POOLING BASED ON DYNAMIC MARKOV MODELING

Nima Sharifimehr and Samira Sadaoui
Department of Computer Science, University Of Regina, Regina, Canada

Keywords: Object pool service, markov model, prediction, automatic tuning, workload modeling, enterprise applications.

Abstract: One of the most challenging concerns in the development of enterprise software systems is how to manage effectively and efficiently available resources. Object pooling service as a resource management facility significantly improves the performance of application servers. However, tuning object pool services is a complicated task that we address here through a predictive automatic approach. Based on dynamic markov models, which capture high-order temporal dependencies and locally optimize the required length of memory, we find patterns across object invocations that can be used for prediction purposes. Subsequently, we propose an effective automatic tuning solution, with reasonable time costs, which takes advantage of past and future information about activities of object pool services. Afterwards, we present experimental results which demonstrate the scalability and effectiveness of our novel tuning solution, namely predictive automatic tuning service.

1 INTRODUCTION

One of the most challenging concerns in the development of enterprise software systems is how to manage effectively and efficiently available resources and realize user requirements. Resource management must be applied across both low-level resources (i.e., CPU time, memory usage) and high-level resources (i.e., database connections) (Jordan et al., 2004). Consequently, to improve the performance of resource management, applicable techniques, such as resource pooling (Kircher and Jain, 2004) and distributed resource allocation (Raman et al., 2003) have been introduced. Here we are interested in resource pooling which not only improves scalability through sharing the cost of resource initialization but also allows accurate tuning of memory usage (Crawford and Kaplan, 2003). Enterprise application servers (Oberle et al., 2004) as sophisticated middlewares significantly take advantage of this technique to accelerate access to resources such as persistent resources, database connections and execution threads. In particular, object-oriented application servers provide object pool services as a special case of resource pooling. For example, IBM WebSphere as a J2EE application server prepares object pool service for entity beans (Anderson and Anderson, 2002).

However, object pool service tuning is a complicated task which has not been addressed precisely. Each object pool service contains a set of object pools whose sizes should be tuned efficiently. Manual tuning of object pools is extremely diffcult, and because of its static nature it can not adapt itself with the changing characteristics of workloads over time (Sullivan et al., 2004). In contrast, automatic tuning can be developed in a dynamic way that is adaptive considering the variable nature of workloads. A traditional automatic tuning solution works only according to past information (Barrera, 1993) and ignores the future information that can be extracted from the workload model.

In this paper, we propose a novel solution, namely Predictive Automatic Tuning Service (PATS), for object pooling. PATS takes into consideration past and future information. In other words, we show that tuning through prediction of object behaviors achieves higher performance comparing to traditional approaches. Different types of Markov models have been used to study stochastic processes and it is shown that they are efficient tools for modeling and

Sharifimehr N. and Sadaoui S. (2007).
A PREDICTIVE AUTOMATIC TUNING SERVICE FOR OBJECT POOLING BASED ON DYNAMIC MARKOV MODELING.
In *Proceedings of the Second International Conference on Software and Data Technologies - Volume ISDM/WsEHST/DC*, pages 38-45
DOI: 10.5220/0001329700380045

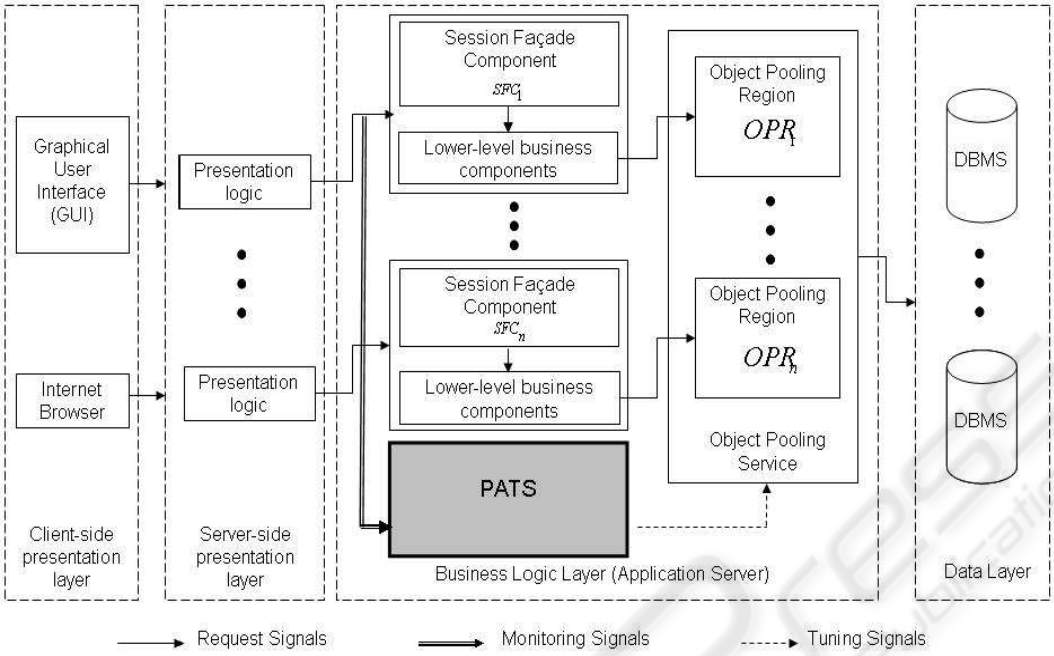

Figure 1: Integration of PATS into the middleware-based enterprise application architecture. This proposed integration approach lets PATS monitor requests sent from each client to different Session Facade Components (SFCs). Monitoring incoming requests is a prerequisite to extract clients behavioral patterns. On the other hand, placing PATS inside the business logic layer is a necessity to be able to send tuning signals to the object pooling service.

predicting users' behavior (Deshpande and Karypis, 2004). Thus, here for prediction purposes, we use Dynamic Markov Modeling (DMM) (Cormack and Horspool, 1987). DMM, which has been introduced in Dynamic Markov Compression (DMC) method (Cormack and Horspool, 1987), is a variant of Variable Length Markov Model (VLMM) (Stefanov et al., 2005). Thanks to DMM, we model the sequence of callings on different objects which use a given object pool service. Through this modeling process, we find patterns across object invocations that can be used as a prediction tool. PATS is fully implemented with Java (a total of 2,000 lines of code). In this paper, we conduct several experimentations to demonstrate the effectiveness of PATS in practice.

The rest of this paper is structured as follows. Section 2 describes how PATS can be integrated into enterprise applications. Section 3 introduces our behavior prediction approach based on DMM. Section 4 explores the detailed design of our automatic tuning approach. Section 5 illustrates the results of workload simulation that evaluates the performance of PATS. Finally, Section 6 concludes the paper with some future directions.

2 PATS INTEGRATION INTO ENTERPRISE APPLICATIONS

The integration of PATS into enterprise applications is an important issue that we address here (cf. Figure 1). Most current enterprise applications are composed of four logical layers (Ebner et al., 2000): client-side and server-side presentation layers, business logic layer and data layer. In middleware-based enterprise applications, the business logic layer is generally implemented inside an application server. This layer is composed of many business components which interact with object pools. However, many business processes involve complex manipulation of these business components. Therefore, Session Facade Pattern (Alur et al., 2001) should be used as a higher-level business abstraction that encapsulates interactions among lower-level business components. In other words, clients must only access application features through Session Facade Components (SFCs) which implement Session Facade Pattern.

According to our integration approach, object pooling services must assign a specific part of the object pool, namely Object Pooling Region (OPR), with a specific size and replacement policy to each SFC. So, objects which are being manipulated by

each SFC can use assigned regions for object pooling purposes. The determination of replacement policy for each OPR has been already successfully addressed by others (Guo and Solihin, 2006). In this paper, we focus exclusively on tuning the size of each OPR. This is a necessary task that significantly affects the applicability of object pooling service, and increases the performance of enterprise applications.

As shown in Figure 2, PATS is composed of three major components: (a) monitoring interface which passively receives incoming calls to SFCs; (b) clients' behavior pattern modeling engine which builds each client's behavior pattern according to its callings on SFCs; and (c) tuning analyzer which decides about the size of each OPR according to built behavior patterns.

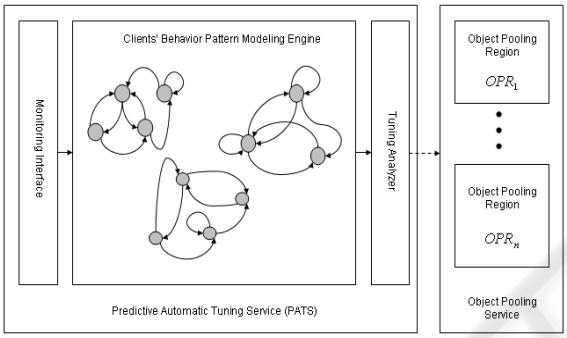

Figure 2: The internal architecture of PATS. Monitoring interface component provides the clients' behavior pattern modeling engine with incoming requests from each client. Then this engine according to the sequences of incoming requests, builds and updates the behavior pattern of each client as a separate markov model. The tuning analyzer component uses these markov models to generate efficient tuning signals which will be sent to the associated object pooling service.

As our tuning solution, PATS monitors requests sent from each client to SFCs. In a real-time manner, it builds and updates the dynamic behavioral pattern of each client according to requests being sent to SFCs. Using those behavioral patterns, we predict the future workloads which are more probable to be sent to different SFCs. Through the prediction of workloads on each SFC, our proposed tuning analyzer dynamically tunes the size of available OPRs in the object pooling service. However, the object pooling service is responsible to apply new size restrictions defined by the tuning analyzer (i.e. releasing extra objects loaded to each OPR). Also, the tuning analyzer reports the new size of each OPR in terms of a fraction of the total size of memory available for object pooling purposes. In other words, the size of objects loaded to each OPR does not affect the tuning computation process. In the next section, we present our prediction approach which is based on DMM.

3 BEHAVIOR PREDICTION

We are interested in building models of behavior which are able to support the prediction of future actions. We define the sequence of callings each client sends to different SFCs as its behavior. Markov model as a simple, yet efficient prediction mechanism is widely used to capture the sequential dependence and constraints of behavioral patterns (Galata et al., 1999). According to the definition (Eirinaki et al., 2005), the n-order Markov model is a directed graph with attached probabilities to each of its edges. The transition probabilities of each state in this graph only depend on 'n' previous states.

Using a fixed order Markov model needs us to know the length of required memory for prediction (Galata et al., 1999). However, because of the dynamic and variable nature of the workload that we are going to model (i.e., the requests being sent to SFCs by each client), using fixed order Markov models is not applicable. In contrast, a Variable Length Markov Model (VLMM) (Stefanov et al., 2005) not only captures high-order temporal dependencies but also locally optimizes the required length of memory (Galata et al., 1999).

Dynamic Markov Compression (DMC) (Cormack and Horspool, 1987) has taken advantage of VLMM to figure out the probabilities of occurrences of future symbols within compressing/decompressing data streams. Here we replace the symbols of data stream with the calling sequence that each client sends to different SFCs, and using DMC approach we build a Dynamic Markov Model (DMM) to predict the future SFC callings (cf. Figure 3). So, our prediction engine updates Markov models as follows:

1. Receives the next SFC request from monitoring interface component.
2. Finds the target Markov model according to the identity of client.
3. Updates the target Markov model using DMC approach.

And for prediction purposes, we extract the required client's associated model from the prediction engine. The probability of each transition from the current state shows the probability of calling on the attached SFC to that transition. Therefore, the number of transitions from each state of our Markov models equals to the total number of SFCs associated with the object pool service.

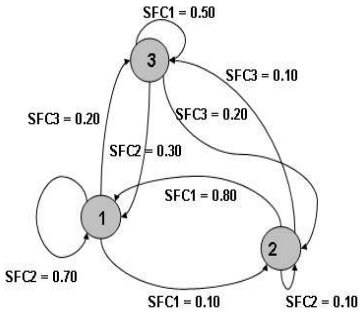

Figure 3: A sample DMM built according to calling sequences sent to SFCs. i.e. when the prediction engine is in state 1, then it will be aware of the fact that the probabilities of incoming calls for SFC_1, SFC_2, and SFC_3 are 10%, 70%, and 20% respectively. Consequently, when the prediction engine sees the next call is SFC_x, it moves on the edge that is labeled SFC_x. Then, the probabilities for next incoming calls will be extracted from the new current state of the engine.

4 AUTOMATIC TUNING

In PATS architecture, tuning analyzer component is responsible for determining the optimal size of each OPR for a given state of the system. This tuner as an effective automatic tuning solution (Sullivan, 2003) meets the following design criteria with reasonable time costs: running without human intervention, adapting to workload variations, and responding to previously unseen workloads.

To tune the size of each OPR, we must figure out the probable workloads which are going to be generated by each client for a SFC connected to the given OPR. For this purpose, we use the associated Markov model with each client. Each Markov model adapts itself with workload changes, so our tuner works considering workload variations. Also we can extract behavior patterns from each Markov model, so our tuner is applicable for unseen workloads. On the other hand, because all the parameters of each Markov model is being tuned automatically, no human interaction is required. Subsequently, we explain the details of our tuning approach to show its accuracy and time costs.

To predict the most probable workloads which are going to be generated by each client, we can use two different approaches with different time costs:

1. Finding the most probable path (i.e., a SFC calling sequence) with a specific length T, for a given Markov model from the current state. This requires to solve the following problem: given a Markov model λ composed of states, each of them having N transitions (i.e. the total number of available SFCs), how do we choose a calling sequence $S = S_1, S_2, ..., S_T$ so that $P(S|\lambda)$, the probability of occurence of the calling sequence with length T, is maximized. As this search space contains N^T different solutions, searching all of the space results in a time complexity of $O(N^T)$. However, we can use greedy algorithms (Cormen et al., 2001) which finds an optimum solution, not the best, with time complexity of $O(N * T)$. This latter in each state chooses the transition with the highest probability, moves to the next state using that transition and repeats this process T times. Although the greedy algorithm is more efficient than the non-greedy one, it suffers from a major issue: when it arrives at a state whose transition with the highest probability is connected to the same state, it does not move to other states anymore and ignores the random nature of those transitions probabilities.
2. Generating a random workload, with a specific length T, according to a given Markov model λ composed of states each of them having N transitions (i.e., the total number of available SFCs). This approach uses an algorithm to generate random numbers (i.e., numbers which are mapped to SFCs) with given probability distributions (i.e., probability distribution of SFC callings from each state of the given Markov model.). There are techniques available for this purpose such as the alias method proposed by Walker (Westlund and Meyer, 2002) with initialization time complexity of $O(N * log(N))$ and generation time complexity of $O(1)$. However, there is also a variation of Walker's original model by Vose (Vose, 1991) whose initialization time complexity is linear $O(N)$ and generation time is the same $O(1)$. Consequently, exploiting this latter to generate a random workload with length T requires $O(N + T)$ time.

According to the time costs of the two approaches mentioned above and the real-time nature of request processing in application servers, we use the second approach which offers a linear time complexity to predict the future workload of each client. Subsequently, using the predicted workloads of all clients, we can predict the Future Workload Ratio (FWR) of each OPR as follows:

$$FWR(OPR_I) = \frac{\sum_{c=1}^{C} seqn_c(OPR_I)}{C * T}$$

where I is the index of an OPR, C the total number of clients, T the length of predicted workload

sequences for each client, and $seqn_u(OPR_I)$ the total number of requests within predicted workload sequence which is going to be sent to OPR_I by client c.

Although now we can use FWRs as future information to tune the size of OPRs, we need to take into consideration past information that give us a feedback about past efficiency of the tuning service. Here we propose a variation of IPCM (Sadaoui and Sharifimehr, 2006) which tunes the size of object pools according to the past information. IPCM calculates the size of each OPR according to its past information including Hit Ratio[1] (HR) and Activity Ratio[2] (AR) through the following formulas:

$$\begin{aligned} BaseSize(OPR_I) &= AR(OPR_I) * C \\ &- HR(OPR_I) * AR(OPR_I) * C \end{aligned}$$

$$\begin{aligned} Size(OPR_I) &= BaseSize(OPR_I) \\ &+ \frac{\sum_{j=1}^{N} HR(OPR_I) * AR(OPR_I) * C}{N} \end{aligned}$$

where I is the index of an OPR, AR a function which returns the activity ratio of a given OPR, HR a function which returns the hit ratio of OPR, C the total size of the object pool, and N the number of OPRs available in the object pool.

In fact IPCM relies on hit ratios to avoid the occurence of starvation for each OPR, and uses activity ratios to divide object pool capacity in a fair manner. However, both of these two factors are related to the past information gathered from object pool activities. Therefore, we define a new concept namely Workload Ratio (WR) which contains both Future Workload Ratio (FWR) and activity ratio which from now on we call it Past Workload Ratio (PWR):

$$WR(OPR_I) = \frac{\lambda * PWR(OPR_I) + \omega * FWR(OPR_I)}{\lambda + \omega}$$

where λ and ω are parameters which separately tune the effect of FWR and PWR on the total amount of WR. Generally, we set equal values for both λ and ω which means FWR and PWR have equal influence on the calculation of WR. Subsequently, we replace AR with WR in IPCM's formulas as follows:

$$\begin{aligned} BaseSize(OPR_I) &= WR(OPR_I) * C \\ &- HR(OPR_I) * WR(OPR_I) * C \end{aligned}$$

$$\begin{aligned} Size(OPR_I) &= BaseSize(OPR_I) \\ &+ \frac{\sum_{j=1}^{N} HR(OPR_I) * WR(OPR_I) * C}{N} \end{aligned}$$

Therefore, according to these new formulas, we consider both past and future information to figure out the size of each OPR. However, in order to use these formulas efficiently we must address an important issue. As a matter of fact, we need to determine an appropriate interval between tuning of OPRs. For this sake, at first we evaluate the performance of our workload prediction in terms of Root Mean Square Error (RMSE):

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (WR_i - WR_i')^2}{N}}$$

Where N is the number of OPRs available in the object pool, WR_i the predicted Workload Ratio for SFC_i, and WR_i' the calculated Workload Ratio according to the generated workload during the last time window of workload measurement. Then, we use a tuning trigger based on a RMSE threshold that automatically runs the tuning process on OPRs whenever the calculated value of RMSE is bigger than a specified threshold. In this way, we make sure that PATS is working with a managed level of error.

5 EVALUATION RESULTS

We conduct our experimentations using synthetic workload models defined by RUBiS (Cecchet et al., 2002) to evaluate both scalability and performance of PATS. These workload models are designed according to standards introduced in TPC-W (Garcia and Garcia, 2003) which model an online bookstore (see clauses 5.3.1.1. and 6.2.1.2 of the TPC-W v1.6 specification (Council, 2001)). We develop an application with SUN's JDK 1.5.0_08 containing 15 SFCs based on RUBiS proposed benchmark architecture. We use a machine equipped with PIV 2.80 GHz CPU, 1 GB of RAM. Also we use MSSQL Server 2000 as the backend DBMS running on MS Windows Server 2003 Service Pack 1.

To evaluate the performance of PATS, we measure the average size of OPRs for PATS, IPCM, and static approach (i.e. the size of all OPRs are equal and fixed.). We carry out this evaluation for different numbers of clients generating different types of workloads for several times. Figure 4 shows the result of this experiment for 10 clients (cf. Figure 4.(a)), 20 clients (cf. Figure 4.(b)), 50 clients (cf. Figure 4.(c)),

[1]The percentage of all accesses that are satisfied by the loaded objects into an OPR.

[2]The percentage of all accesses that are processed through a specific OPR.

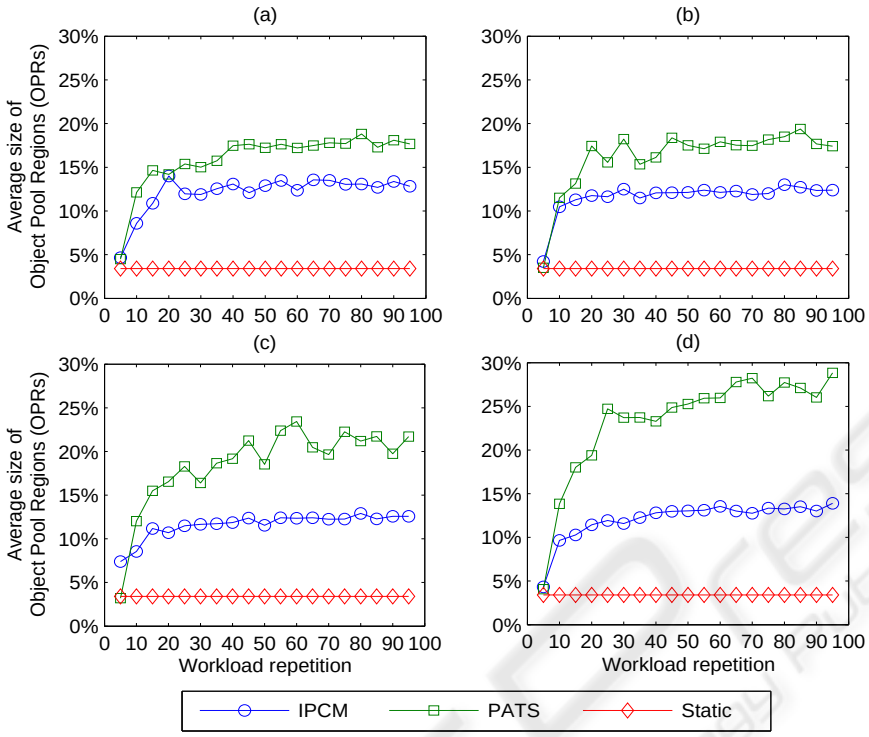

Figure 4: Comparing the average size of OPRs for PATS, IPCM, and static approach for different numbers of clients generating different types of workloads for several times. Diagrams a, b, c and d show the result of this experiment for 10, 20, 50 and 100 clients respectively. In each test case, we keep the number of clients fixed and only change the number of workload repitition. As shown the efficiency of PATS increases after more repitition of workload models and takes over two other approaches significantly.

and 100 clients (cf. Figure 4.(d)). As shown the efficiency of PATS increases by passing the time. The period of time in which the efficiency of PATS is increasing is called the learning period. In other words, PATS needs time to learn the workload model of each client. So, after passing the learning period which results in DMMs with a stable prediction accuracy, the performance of PATS takes over two other approaches significantly.

To evaluate the required learning period for different workload models, we generate workloads based on same workload models for repeated times and measure the mean true prediction after each repetition. True prediction means that the prediction engine predicted the next incoming call was going to be sent to SFC_x and the next incoming call did go to the SFC_x. The result of this experiment for twelve different workload models is shown in Figure 5. With respect to acquired results, it can be seen that the accuracy of prediction in most of test cases gets stabilized after ten repetitions.

To measure the relationship between the maximum size of DMMs (i.e. the maximum number of available nodes in DMMs) and the prediction accuracy of PATS, we generate workloads based on the same workload models with different allowed maximum size of DMMs. Figure 6 shows the impact of increasing the size of DMMs on the prediction accuracy of PATS for twelve different workload models. Interestingly, increasing the size of DMMs improves the prediction accuracy only for a while. In other words, the prediction accuracy of PATS becomes stable with a limited maximum size of DMMs.

6 CONCLUSIONS AND PROSPECTS

In this paper, we have presented a novel predictive automatic tuning service (PATS) for object pool services. Integration of PATS into enterprise applications has been considered by offering detailed architectural requirements. Our proposed approach is based on Dynamic Markov Modeling for the prediction of future

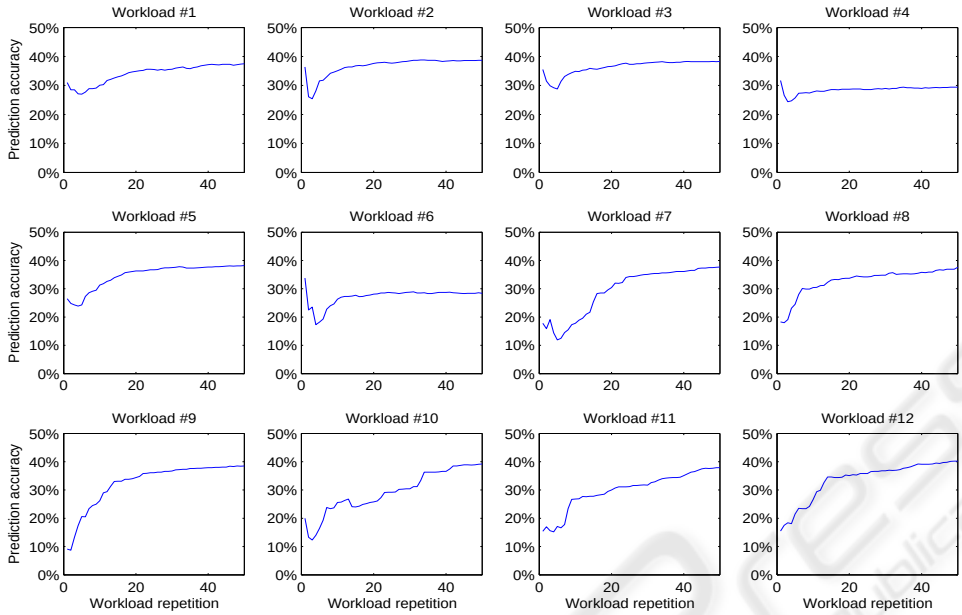

Figure 5: The prediction accuracy versus the number of workload repetition for twelve different workload models. In each test case, a specific workload model has been repeated 50 times. During this repetition, the learner Markov model has been evolving to prepare better prediction accuracy. The prediction accuracy shows the percentage of true predictions made by each Markov model. As shown in this figure, the prediction accuracy of most of Markov models have been stabilized between 30% and 40% only after 10 repetitions.

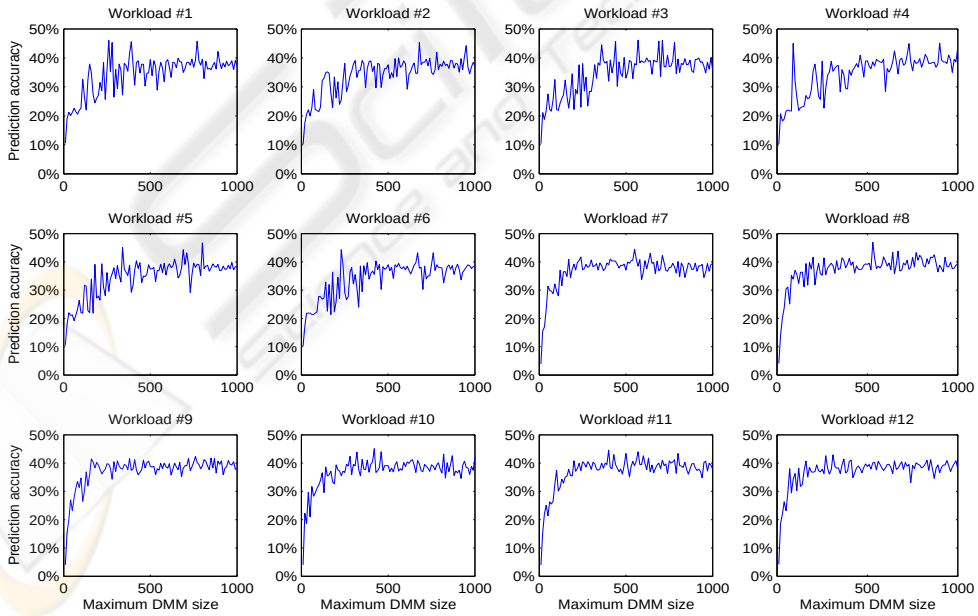

Figure 6: The prediction accuracy versus the maximum size of a DMM (i.e. the maximum number of available nodes in a DMM) for twelve different workload models. As shown in figure above, increasing the size of DMMs improves the prediction accuracy only for a while. In other words, using only a limited amount of memory space, Morkov models prepare a proper level of prediction accuracy which is between 30% and 40%.

probable workloads. We have elaborated our tuning approach and discussed its time costs. Finally, using RUBiS benchmark workloads, we have conducted some experiments and evaluations. The results of our evaluations prove the scalability and effectiveness of using PATS in enterprise applications. The next goal of our research is to investigate how context-oriented approaches can be used to precise the workload modeling. Another goal is to provide the capability to consider the descriptive characteristics of workloads for accelerating the process of workload model learning.

REFERENCES

Alur, D., Malks, D., and Crupi, J. (2001). *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Anderson, P. and Anderson, G. (2002). *Enterprise JavaBeans Components Architecture: Designing and Coding Enterprise Applications*. Prentice Hall Professional Technical Reference.

Barrera, J. S. (1993). Self-tuning systems software. In *Proc. Fourth Workshop on Workstation Operating Systems*, pages 194–197.

Cecchet, E., Marguerite, J., and Zwaenepoel, W. (2002). Performance and scalability of ejb applications. In *OOPSLA '02: Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 246–261, New York, NY, USA. ACM Press.

Cormack, G. V. and Horspool, R. N. S. (1987). Data compression using dynamic markov modelling. *The Computer Journal*, 30(6):541–550.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to algorithms*. MIT Press, Cambridge, MA, USA.

Council, T. P. P. (2001). *TPC Benchmark W, Standard Specification*.

Crawford, W. and Kaplan, J. (2003). *J2EE Design Patterns*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Deshpande, M. and Karypis, G. (2004). Selective markov models for predicting web page accesses. *ACM Trans. Inter. Tech.*, 4(2):163–184.

Ebner, E., Shao, W., and Tsai, W.-T. (2000). The five-module framework for internet application development. *ACM Comput. Surv.*, 32(1es):40.

Eirinaki, M., Vazirgiannis, M., and Kapogiannis, D. (2005). Web path recommendations based on page ranking and markov models. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 2–9, New York, NY, USA. ACM Press.

Galata, A., Johnson, N., and Hogg, D. (1999). Learning behaviour models of human activities. In *Proc. British Mashine Vision Conference (BMVC'99)*, pages 12–22.

Garcia, D. F. and Garcia, J. (2003). Tpc-w e-commerce benchmark evaluation. *Computer*, 36(2):42–48.

Guo, F. and Solihin, Y. (2006). An analytical model for cache replacement policy performance. In *SIGMETRICS '06/Performance '06: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 228–239, New York, NY, USA. ACM Press.

Jordan, M., Czajkowski, G., Kouklinski, K., and Skinner, G. (2004). Extending a j2ee server with dynamic and flexible resource management. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 439–458, New York, NY, USA. Springer-Verlag New York, Inc.

Kircher, M. and Jain, P. (2004). *Pattern-Oriented Software Architecture: Patterns for Resource Management*. John Wiley & Sons.

Oberle, D., Eberhart, A., Staab, S., and Volz, R. (2004). Developing and managing software components in an ontology-based application server. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 459–477, New York, NY, USA. Springer-Verlag New York, Inc.

Raman, R., Livny, M., and Solomon, M. (2003). Policy driven heterogeneous resource co-allocation with gangmatching. *hpdc*, 00:80.

Sadaoui, S. and Sharifimehr, N. (2006). A novel object pool service for distributed systems. In *The 8th International Symposium on Distributed Objects and Applications*, New York, NY, USA. Springer Verlag New York, Inc.

Stefanov, N., Galata, A., and Hubbold, R. (2005). Real-time hand tracking with variable-length markov models of behaviour. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*, page 73, Washington, DC, USA. IEEE Computer Society.

Sullivan, D. G. (2003). *Using probabilistic reasoning to automate software tuning*. PhD thesis, Harvard University Cambridge, Massachusetts. Adviser-Margo I. Seltzer.

Sullivan, D. G., Seltzer, M. I., and Pfeffer, A. (2004). Using probabilistic reasoning to automate software tuning. In *SIGMETRICS '04/Performance '04: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 404–405, New York, NY, USA. ACM Press.

Vose, M. D. (1991). A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. Softw. Eng.*, 17(9):972–975.

Westlund, H. B. and Meyer, G. W. (2002). A brdf database employing the beard-maxwell reflection model. In *Graphics Interface*, pages 189–201.