Access Control Caching Strategies

An Empirical Evaluation

Mathias Kohler
SAP Research
Vincenz-Priessnitz-Str. 1
76131 Karlsruhe
Germany
mathias.kohler@sap.com

Achim D. Brucker
SAP Research
Vincenz-Priessnitz-Str. 1
76131 Karlsruhe
Germany
achim.brucker@sap.com

ABSTRACT

Modern enterprise systems comprise a fine-grained enforcement of complex access control policies. Consequently, the efficient evaluation of security policies is a significant factor for the overall system performance. Moreover, modern enterprise systems are inherently based on process and workflow models. These models enable new approaches for improving the performance of security evaluations.

Caching is widely used for improving the performance and the reliability of systems. The dynamic nature of today's workflow systems, both in terms of changing workflows and in terms of dynamic security policies impose particular challenges on the caching of access control decisions.

We present a caching strategy that exploits business process models for avoiding cache misses. Moreover, we provide a detailed performance analysis of different caching strategies for static and dynamic aspects of access control policies, providing the required metrics for informed design decisions.

Categories and Subject Descriptors

D.4.6 [**Software**]: Operating Systems—*Security and Protection*

General Terms

Security, Performance, Access Control

Keywords

Business Process Security, Access Control, Caching

1. INTRODUCTION

Modern business process-driven enterprise systems, e. g., for enterprise resource planning (ERP) or customer relationship management (CRM), comprise, on different levels, a large number of access control enforcement points (PEPs). As such, the time needed for evaluating access control decisions may significantly influence a user's experience in using such systems. While delays of 100 ms are perceived as a minor interruption, delays of more than one second significantly impact on the flow of thoughts [16]. State of the art industrial workflow systems execute hundreds of thousands of business process tasks in parallel. Modern enterprise systems are inherently based on process and workflow models. These model-centric systems enable new approaches for caching strategies improving the performance of, e. g., security evaluations.

Caching access control decisions is one way of increasing the system performance and, thus, minimizing the delays that users observe. There is a wealth of literature discussing both generic (e. g., [5, 11, 15]), i. e., independent of the application area, and access control specific (e. g., [2, 4, 6, 12, 20]) caching strategies. None of them, however, makes use of the underlying process and workflow models. ProActive Caching [13, 14], in contrast, is to our knowledge the first workflow specific caching strategy which exploits the business process and workflow models for avoiding cache misses.

Metrics are a prerequisite for comparing different caching strategies and, thus, for allowing system designers to choose the optimal solution for a specific system. Both, the performance and the cache size, in relation to different process types and security policy types, provide metrics for comparing access control caching strategies. In this paper, we present and analyze ProActive Caching according to performance improvements and provide a detailed comparison with other approaches for caching role-based and dynamic access control decisions in business process-driven systems. In particular, we compare generic caching strategies against caching strategies developed to specifically cache access control decisions, i. e., $\text{SAAM}_{\text{RBAC}}$ and ProActive Caching. $\text{SAAM}_{\text{RBAC}}$ [20] is a caching strategy developed for systems which rely on role-based access control (RBAC) [17]. Moreover, we present a novel combination of ProActive Caching with the other caching strategies.

Our contributions are three-fold: Firstly, we analyze and present the performance tests for several access control caching strategies. Secondly, we present a novel, two-leveled caching strategy combining existing caching strategies for dynamic access control policies with strategies for static access control policies. Thirdly, we present a generic caching architecture for caching access control decisions in business process-driven systems, which is well suited for performing above mentioned performance measurements.

.

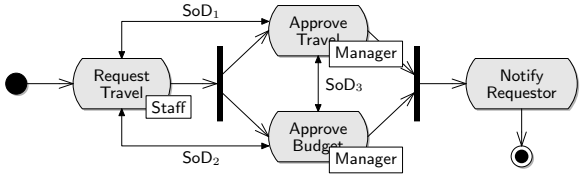


Figure 1: A simple travel approval process

Figure 2: An exemplary General Worklist (GWL).

2. PROCESS EXECUTION

Modern enterprise systems are usually process-driven, i. e., the behavior of the system is based on executing process models by a workflow engine. A workflow engine, e. g., JBoss jBPM [18], executes processes based on their abstract model. Fig. 1 illustrates a simple example of a process for approving travel requests: a staff member may issue a travel request, which needs to be approved by her manager and a manager approving the travel budget. Afterwards, the requesting user is notified if her request is granted or not. The execution of a task may be user-centric or automatic. In the later case, the system executes a task without user interaction; in the former case, a human is responsible for executing a task, i. e., the user has to claim the task to work it off.

The *General Worklist* (GWL), also called *Universal Worklist* (UWL), is the main interaction point for user-centric process execution. Fig. 2 shows a GWL presenting the list of tasks to the users from which she selects the tasks she is able to claim. Usually, a user may only claim a small fraction of all tasks available in the system. Tasks may not be claimable based on a lack of required access rights. For instance, assume a system allowing for enforcing access control restrictions based on both, a hierarchical role-based access control (RBAC) [17] model and dynamic separation of duty requirements (DSoD). RBAC allows to specify roles (i. e., groups of users) that are allowed to execute (claim) a task. We assume two roles "Staff" and "Manager" whereas every "Manager" is also a member of the role "Staff." Thus, in our example (see Fig. 1), every member of the staff can issue a travel request, but only managers can approve them. In fact, each travel request needs to be approved by two managers, one of them taking the responsibility for the budget. Of course, we want to ensure that no user is able to approve his own travel requests. This requires that the requesting user and both approving managers are pairwise different. The evaluation whether a user may perform one of the approval tasks must be evaluated during runtime and depends on the tasks the user already performed.

Dynamic access control constraints require that the ability of a user to claim a task needs to be checked for every task each time the GWL is displayed—as the context enabling a user claiming it might have changed since she entered the GWL the last time. The number of tasks in the GWL mainly depends on the number of process instances running in the system. If 300 process instances are active, each of them having at least one but possibly multiple active tasks would result in more than 300 active task instances. If only 30% of them are in a status ready to be assigned, meaning they would show up in the GWL, access control checks for about 100 tasks must be performed every time a worklist is displayed for a user. An access control check requires about 20 ms to 40 ms which results in a delay of two to four seconds every time the GWL is displayed.

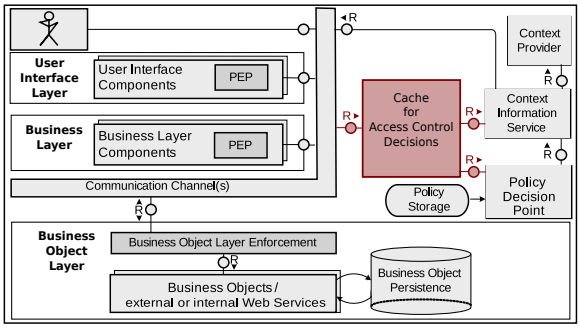


Figure 3: Process-driven systems

3. AN AC CACHING ARCHITECTURE

Modern enterprise systems are multi-layered comprising a user interface (UI) layer, a business layer, and a business object layer (see Fig. 3). A business process management system (BPMS) links all three layers; within service-oriented architectures, however, all of them are directly accessible. Hence, during process execution, access control takes place at various layers. The UI layer contains the GWL and realizes the above described access checks for displaying a worklist. The business layer is responsible for process and task management. Independent from the GWL, it must check user requests according to process and task executions. The business object layer provides encapsulated functionality to back-end systems using business objects. At this level it must be checked, whether a user is allowed to access a respective business object.

In large enterprise systems access control enforcement usually follows the request-response paradigm: a *Policy Decision Point* (PDP) answers the access control requests from one or several *Policy Enforcement Points* (PEP) which in turn enforce these access decisions within the BPMS. Given the three layers, each of them may contain one or more PEPs enforcing decisions from the PDP.

We extend this standard architecture for enterprise systems with a caching component. This component caches access control decisions issued by the PDP for accelerating the evaluation of future access requests. Thus, the caching component acts as a proxy between the different PEPs and the PDP. Furthermore, the caching component may issue access requests to the PDP on its own discretion to realize ProActive Caching. This architecture does not depend on the implemented caching strategy: generic caching strategies (e. g., [5, 11, 15]) can be as easily integrated as access control specific ones (e. g., [2, 4, 6, 12, 14, 20]).

4. PROACTIVE CACHING

The core idea of ProActive Caching (PAC) [13, 14] is to improve the system performance, especially for user-centric tasks, by exploiting the knowledge about possible system behavior (e. g., process and workflow models).

The objective of PAC is to optimize the availability of cache entries by anticipating which access control request evaluations are required during the execution of a business process. This allows us to pre-evaluate access requests and store the respective access decisions such that they are available when needed and, thus, even first-time queries can be answered from the cache. In its essence, PAC uses the process and workflow models to anticipate and pre-evaluate all access control requests occurring during the execution of a business process and store the respective access decisions in a cache (The heuristics for updating cache entries are described in [14].). Recall our travel approval process (see Fig. 1) and assume that Alice is requesting a travel (i. e., she claims an instance of the task "Request Travel"). In this situation, we can already pre-evaluate, for all potential users, access requests for the two immediate upcoming approval tasks, i. e., "Approve Travel" and "Approve Budget." Recall, opening the GWL requires access checks for all available tasks in the system. If a manager Bob accesses the GWL to claim "Approve Travel," the respective access decisions for showing the tasks he may claim are already available. This reduces the time for displaying the worklist from several seconds to a few hundred milliseconds.

Furthermore, PAC enables to cache access decisions based on policy constraints which rely on dynamic context information (e. g., time of day, execution history). The evaluation of dynamic constraints (e. g., DSoD) depends on the current system state, i. e., in such scenarios the PDP is not stateless. The problem with cached access decisions based on dynamic constraints is that they may become invalid over time. This means the same access request evaluation might have a different result if the context information changed in between the two requests, making already cached access decisions invalid and obsolete. Overall, the update strategies of PAC ensure (see [14] for details) that, at any time, all cached access control entries are valid with respect to both the active access control policy and current system state.

In our example, Bob may not perform both approval tasks. Hence, if Bob already performed "Approve Budget," any cached access decision granting Bob to perform the second approval task "Approve Travel" becomes invalid. ProActive Caching deals with invalid decisions by constantly monitoring the status of the BPMS and revoking or updating them pro-actively if necessary. With the assignment of Bob to "Approve Budget" it is clear all function calls on business objects to execute the task will be done on Bob's behalf and responsibility. Hence, with the assignment, all access requests for function calls on the business objects are pre-evaluated such that they are available upon their request.

5. PROCESS EXECUTION CACHING

5.1 A Classification of Caching Approaches

Caching approaches for business process-driven systems can be categorized into: 1. strategies that are optimized for a specific caching domain (e. g., caching of access control decisions) versus strategies that are designed for caching arbitrary data and 2. strategies that are optimized for workflow management systems (i. e., inherently exploiting process or workflow models) versus strategies that are designed for arbitrary systems. In this paper, we will discuss the following basic caching approaches for caching access control decisions in business process-driven systems (see Fig. 4):

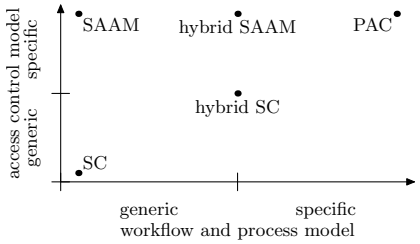


Figure 4: A taxonomy of caching approaches

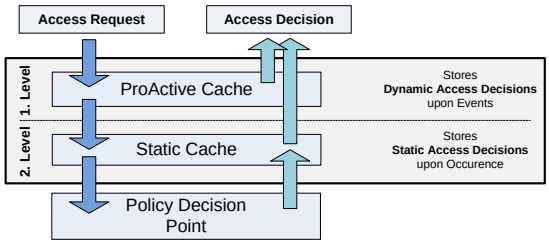


Figure 5: A multi-leveled caching architecture

1. *standard caching* (SC). In a standard cache, each access decision response which is not found in the cache is evaluated by the PDP and its response stored in the cache, optimizing identical requests in the future. As SC was developed as a generic caching strategy, it neither exploits specific features of the access control model used nor of any underlying process models.
2. *secondary and approximate authorization model* (SAAM), in particular, $\text{SAAM}_{\text{RBAC}}$ [20], is optimized for caching decisions for RBAC models. SAAM uses already cached access decisions to infer further access decisions which have not yet been cached. Inferred decisions are called approximate decisions. Hence, also new requests which did not appear yet can possibly be answered by a cache look-up. SAAM exploits specifics of the RBAC access control model but is not tailored for business process-driven systems.
3. *ProActive Caching* (PAC) is especially developed with both process-driven environments and dynamic security requirements in mind. PAC pre-evaluates access decisions according to a caching heuristic based on both the deployed process models and the access control model used. The pre-evaluated access control decisions are stored in the cache.

SC and SAAM are designed for caching static access control decisions. As dynamic access control policies (e. g., DSoD constraints) are required more often [8–10] by business regulations such as Basel II [1], we also present a combination of the different caching approaches, i. e., hybrid caching strategies. They use PAC for the dynamic aspects of the security policy (i. e., as a kind of first-level cache) and either SC or SAAM for the static parts of the policy (i. e., as a kind of second-level cache). In its essence, such hybrid caching strategies can be implemented by using a PAC as proxy which transparently forwards all static requests to the underlying cache optimized for static security

policies. Thus, hybrid caching can be understood as variant of a multi-leveled caching approach using SC or SAAM as first-level cache and PAC as second-level cache (see Fig. 5).

5.2 Experimental Results

We compare the different caching strategies based on our GWL scenario motivated in Sect. 2: we simulate a process-driven enterprise system and measure the time spend in evaluating access control requests that are required for displaying a user's GWL. In more detail, we use an implementation of the architecture discussed in Sect. 3. We configured the system with various process models taken from the set of reference models for SAP's R/3 systems [7]. Notably, we enriched these process models with a role-based security policy and, depending on the experiment, with DSoD constraints. In our experiment, we simulate the user-driven execution of several instances of the different business processes in parallel and measure the time for evaluating all access control requests that are required for displaying the GWL. For a single process execution, we simulate a scenario in which a user calls her GWL, selects (claims) a task to work on and successfully finishes this task. Thereafter, she works on the upcoming tasks until the process is finished.

For this scenario, we consider different numbers of process instances, ranging from 25 to 300 instances executed in parallel. All process instances are based on four different process definitions (having up to 25 tasks) selected from SAP's reference models. Our security configuration comprises 100 users, 20 roles, and about 8000 permissions. For each process definition, we assigned two roles that are allowed to instantiate a process definition and every user is a member of five randomly chosen roles. As in human-centric workflows every task has to be assigned to a user via the worklist, the GWL is at least displayed once for each task instance. For example, for 25 process instances of medium sized process definitions (each having about 7 tasks) results already in at least 175 worklist calls in total. We used this setting for all experiments we report on in this paper and, for all tests, we used a standard server (3.4 GHz Intel Xeon CPU, 8 GiB of RAM).

In the following, we first have a look on static access control requests (comparing no caching, SC, SAAM, and PAC) and, afterwards, we add dynamic access control requests (comparing no cache, SC, SAAM, PAC, and hybrid caching).

5.3 Static Access Control

Fig. 6 summarizes the comparison of executing our scenario using no caching, SC, SAAM, PAC with static access control policies. The average time required for evaluating a single access control request (see Fig. 6a) without caching slightly increases with the number of process instances running in parallel. PAC, in contrary, is independent hereof. Moreover, PAC results in a much faster evaluation of access control requests, always remaining faster than 1 ms (while no caching results in evaluation times between 27 ms and 38 ms). A similar behavior is observed for the average time required for displaying the GWL (see Fig. 6b).

With an increasing number of parallel process instances, both SC and SAAM converge to PAC's low response time. This behavior is due to the fact, that with more processes running in parallel, the likelihood that an access control request is already cached increases. This assumption is verified by the hit rate of the different caching strategies (see Fig. 6c) which converges to 100% for SAAM and SC. Our experiments support the results of Wei et al. [20]: in comparison to SC, SAAM converges significantly faster to the response time of PAC.

Fig. 7 shows the different times for displaying the GWL using *box-plots*: the heights of the fine lines above and below the boxes show the minimal and maximal time required to display a worklist. The boxes start and end with the lower and upper quartiles of the required times. Additionally, the graphs show the experiments' median values, illustrating the maximum time a user has to wait in 50% of all cases.

Displaying the worklist using no caching (see Fig. 7a) takes for 300 process instances almost 8 s in 75% of all cases. As the median is close to the upper box limit, we conclude that only a few cases take significantly longer. In fact, our data shows that in up to 99% of all cases the user has to wait up to 8.5 s.

For SC (see Fig. 7b) a user waits up to 700 ms for the worklist to display. The probability that a user has to wait more than 100 ms, however, decreases with increasing number of process instances. This is substantially faster compared to the implementation without caching (see Fig. 7a) where delays in the range of seconds are normal for the average case.

For SAAM (see Fig. 7c) the maximal time is lower as for SC and does not exceed 250 ms. Especially the cases for 300 process instances are very dense with an upper box limit at about 17 ms to display a worklist. Our data shows that in 93% of all cases the required time remains below 50 ms.

The times for PAC (Fig. 7d) are always less than 16 ms. Comparing the tests starting from 25 up to 300 process instances the average values to display the worklist even increase. This is due to PAC's caching strategy which results in a nearly 100% cache hit rate (cf. Fig. 6c). Hence, nearly all access requests required to display a worklist are answered by the cache (rather than by the PDP) such that the time to display the worklist is the sum of all cache lookups.

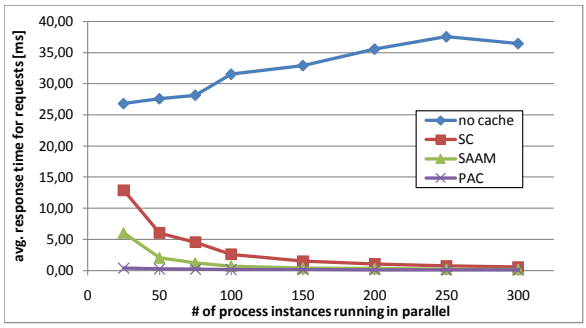
5.3.1 *Discussion.*

The results show that for static environments caching is very effective in reducing the response time for access requests. Notably, even the standard cache for small scenarios already decreases the average access time to half of the time required in implementations without cache.

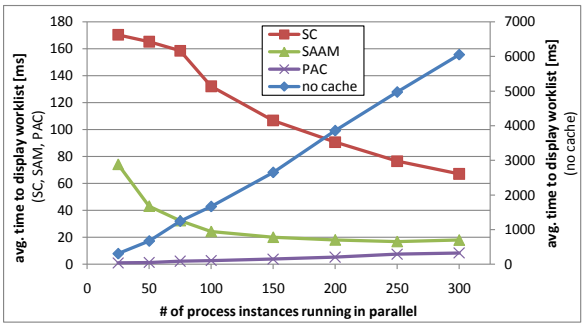
Further, in our experience both caching strategies developed for static environments (standard caching and SAAM) are easy to implement and achieve, already with a comparatively small amount of process instances, a benefit comparable to the PAC. Moreover, standard caching and SAAM require a warm-up phase for filling the cache. During this warm-up phase, the response time of the system may be considerably higher. The maximum values in Fig. 7 also confirm this observation.

While the PAC implementation requires a higher effort, it results in a system which can ensure response times remaining below a critical time of, in our case, 2 ms. This is due to the fact, that PAC does not have a warm-up phase (as it pre-evaluates access requests) and thus, the deviation of times needed for evaluation is small. Albeit, we need to invest additional system resources for pre-computing cache entries that might never be used.

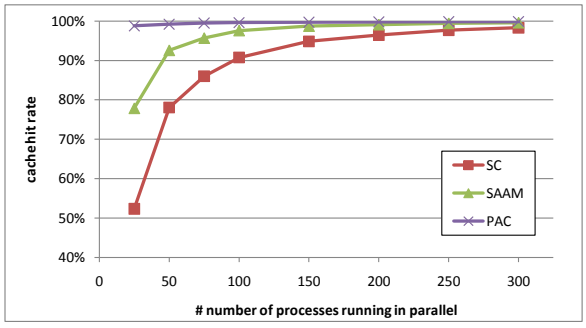
Furthermore, caching comes at a cost: for all caching variants we have to store the cached access control decisions in memory. Fig. 6d illustrates the maximal number of cache en-



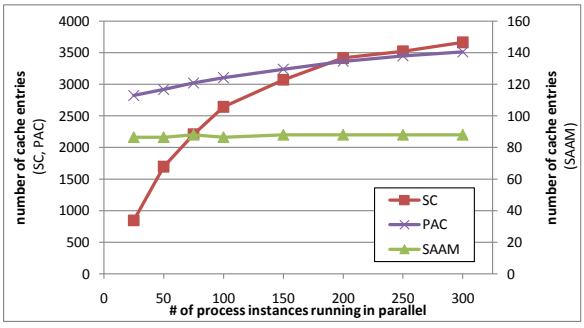
(a) Average access time



(b) Average time for worklist display

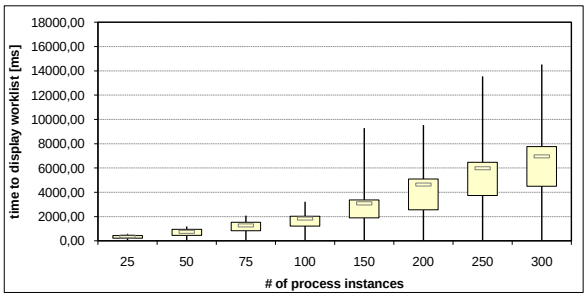


(c) Hit rate

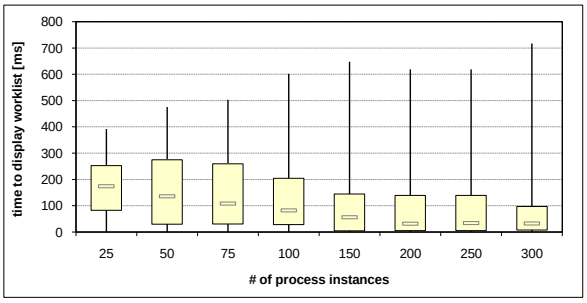


(d) Maximum cache size

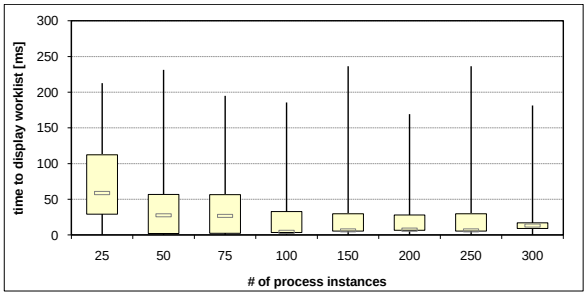
Figure 6: Static Access Control: Overview



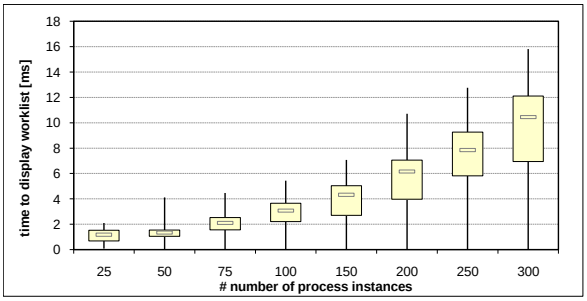
(a) GWL No Cache



(b) GWL SC



(c) GWL SAAM

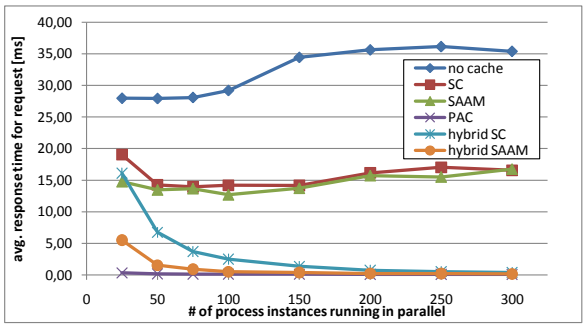


(d) GWL PAC

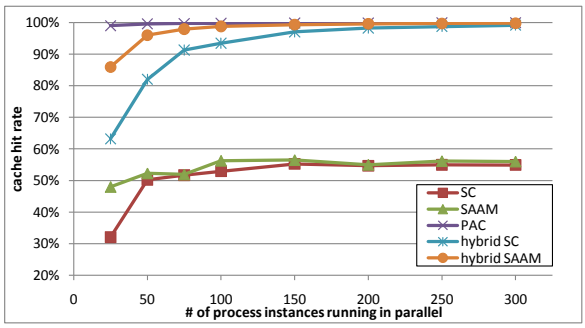
Figure 7: Static Access Control: GWL

tries for SC and PAC (left y-axis) and SAAM (right y-axis). The graph shows that already for small scenarios, the heuristics used by PAC result in pre-evaluating and caching nearly all access control decisions beforehand. In contrast, SAAM requires a much smaller number of cache entries. One has to take into account, however, that this comparison ignores the fact, that the cache entries of SAAM are typically larger than the cache entries of PAC and SC. Measuring the memory consumption reveals that SAAM requires approximately half the memory as PAC or SC for large scenarios.

In conclusion, all caching strategies tested generate a strong impact with respect to a system's response time com-



(a) Average access time



(b) Hitrate

Figure 8: Average access time and hitrate for scenarios with dynamic policies (DSoD)

pared to systems without caching. If a system requires a guaranteed response time PAC seems to be most suited and the additional effort required for PAC seems for to pay off. Moreover, this already proves the additional benefits of exploiting process and workflow models for caching the access on static system resources.

5.4 Dynamic Access Control

In the following, we enrich our scenario with dynamic access control requirements, i. e., policies with DSoD constraints: overall, 40% of the tasks of a process definition are affected by a DSoD constraint. DSoD constraints rely on dynamic context information. A cache storing access decisions which are based on the evaluation of DSoD constraints, hence, needs to be updated if the context changes. PAC supports such cache updates while SC and SAAM do not. Applying SC and SAAM in a dynamic environment requires that no access decisions based on the evaluation of DSoD constraints are cached. Hence, we specifically forward all access requests which require DSoD constraints to be evaluated directly to the PDP for regular evaluation. The information which requests have to be forwarded is available as it is clear which tasks of the used process definitions are affected by an DSoD constraint.

Fig. 8a shows the average time it takes for each caching strategy to evaluate an access request. Similar to the static scenario, the response time for no caching increases with the number of process instances. SC and SAAM tend to stabilize in our experiments at a level around 16 ms. The reason for this behavior is that dynamic access control decisions are not cached and, hence, every dynamic access request must be evaluated by the PDP. The cache hit rate of SC and SAAM (see Fig. 8b) roughly converges against a value equal

# processes	PAC [s]	Hyb. SC [s]	Hyb. SAAM [s]
25	119	26	24
50	127	30	31
75	135	42	41
100	145	44	46
150	161	61	61
200	173	66	68
250	192	70	84
300	214	81	85

Table 1: Comparing cache management efforts

to the probability that an access request is not affected by a DSoD constraint.

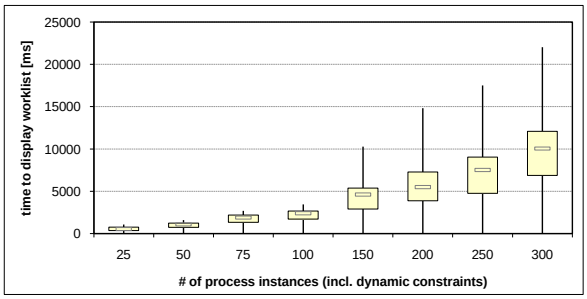
PAC shows a similar behavior as in the static scenario. This, however, comes not for free as the strategy requires that anticipated access requests during a process execution are pre-evaluated, independent whether these cached decisions will be used afterwards. We measured the additional effort required. Tab. 1 illustrates the results, i. e., showing the total time (in seconds) solely required for cache management by pre-evaluating anticipated access requests. The time for cache management for PAC ranges from 119 s to 214 s given the respective parallel execution of 25 to 300 process instances.

The pre-evaluations of PAC include both static and dynamic access requests which causes overhead. To reduce this overhead we introduce hybrid caching. Hybrid caching combines static caching strategies with the proactive strategy (cf. Sect. 5.1). We distinguish between Hybrid SC (combining SC and PAC) and Hybrid SAAM Caching (combining SAAM and PAC). The idea in both cases is that for static access requests the static caching strategies are used (which do not require additional effort for cache management); for the dynamic access requests the cache and cache-update strategies of PAC are used.

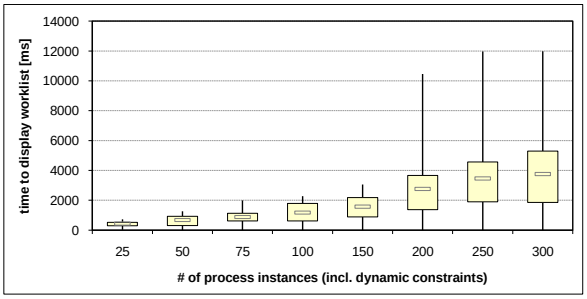
Fig. 8a shows that the static parts of hybrid caches still require a certain time until access requests can directly be answered from the cache. The cache hit rate, however, converges against 100% with increasing number of process instances (see Fig. 8b). This is fact for both cases, combining SC and PAC or SAAM and PAC. Also, it can be seen that given any number of process instances, the hybrid versions are always faster than their static counterparts. Tab. 1 shows that we achieved our main goal for hybrid caching: the reduction of the cache management. The time for cache management for the hybrid caching strategies is required to update its proactive cache for dynamic requests.

The box-plots of Fig. 9 illustrate the different times it takes to display the worklist with caching and, as reference, without caching (see Fig. 9a). The latter shows that displaying the worklist takes significantly more time if compared to static environments. This is the case as the evaluation of dynamic access constraints, i. e., DSoD constraints generally require the additional effort to fetch and evaluate dynamic context information, leading to a longer request evaluation time—compared to requests evaluated based on standard role-based policies.

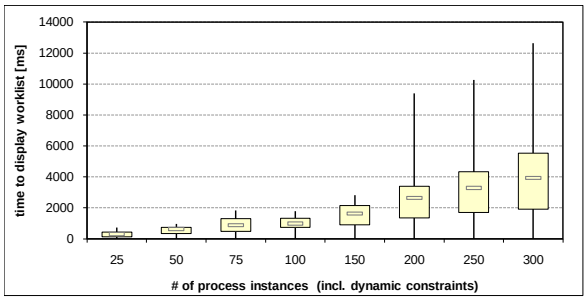
The box-plots for the SC (see Fig. 9b) and SAAM (see Fig. 9c) show that the time to display the worklist increases with the number of instances in a system, rather than decreasing as it is the case in the static environment. This behavior, again, results from the fact, that dynamic access requests are always regularly evaluated. Given 200 and more



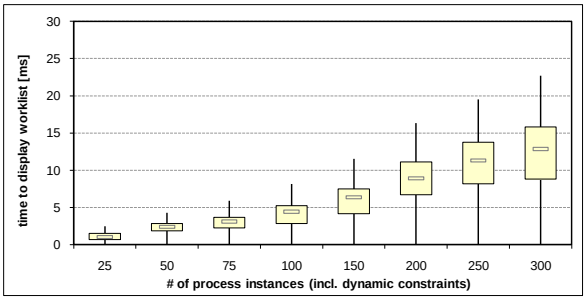
(a) GWL No Cache



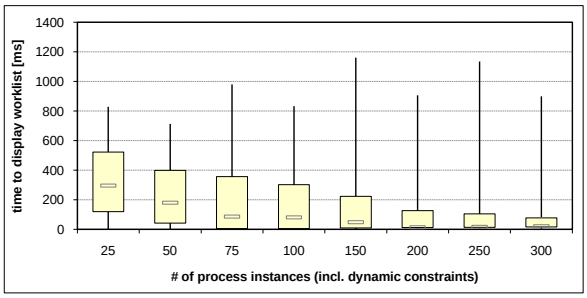
(b) GWL SC



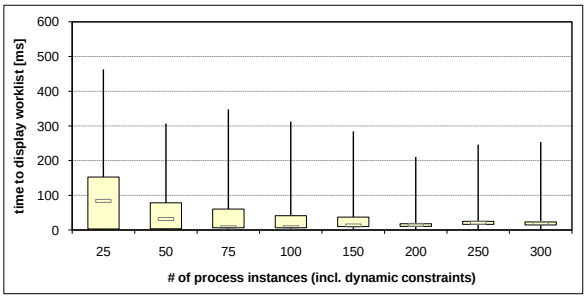
(c) GWL SAAM



(d) GWL PAC



(e) GWL Hybrid SC



(f) GWL Hybrid SAAM

Figure 9: Dynamic Access Control: GWL

process instances the difference between SC and SAAM is very small; the times required for the regular evaluation of dynamic access requests clearly dominate results.

The PAC remains the fastest strategy within the complete tested scenario. The worklists can be displayed after a maximum of 23 ms. The two hybrid caches behave similar compared to SC and SAAM in the static environments (compare Fig. 9b with Fig. 7b and Fig. 9c with Fig. 7c). The box-plots show, however, that the static parts of the caches require time to fill the cache. This warm-up period leads to possible times above 1 s in case of Hybrid SC and above 300 ms in case of Hybrid SAAM. In both cases, however, those spikes are quite rare as in 90% of all cases the time to display the worklist lies below 230 ms and 50 ms.

5.4.1 Discussion.

Overall, our experiments show that in dynamic environments the use of process and workflow models can increases the overall system performance. In particular, caching approaches that exploit these models perform significantly better than workflow and process model independent approaches such as SC or SAAM. In more detail, already a considerably low amount of dynamic constraints (in our scenarios only 40% of the total resources were affected) result in a significant impact on response times. Dynamic access requests cannot be cached and, hence, lead to increased cache misses. We expect that in future, the general amount of dynamic security constraints will rather increase than decrease. An increasing amount of dynamic constraints would lead to further increasing response times if no caching or static caching strategies are used.

In contrast, PAC can still ensure that the response times remain below a certain critical time, given the system allows to detect context changes which can be used to update the cache. In our experience, the additional effort PAC requires for its cache management sums up to, e. g., 214 s for 300 process instances (cf. Tab. 1). Of special interest are the results for two-level caching architectures combining both static and dynamic caching strategies. Using the PAC update strategy for dynamic access decisions as first level cache enables the use of a static caching strategies in dynamic environments as second level cache. Also in this case, however, the combined caches have a warm-up phase leading to situations where the time to display the worklist is up to ten times above the average time of the same scenario.

Based on the results for dynamic environments we can conclude that for systems requiring a guaranteed response time, again PAC should be preferred and the additional effort required remains within a reasonable range. In all other cases a mixture between established caching strategies for static environments combined with the update strategy of the PAC seem the preferred choice. Again, if a combination with the tested SAAM strategy is considered, the systems static policies should be based on RBAC.

6. CONCLUSION AND FUTURE WORK

Although there is a large body of literature concerned with improving the access control decisions in IT systems in general, and using caching approaches in particular (e. g., [2, 4, 6, 12, 13]), none of them compares the different approaches on a common ground. While Turkmen and Crispo [19] compare the performance of different PDP implementations, our work, is to our knowledge, the first comparison of different caching strategies on a uniform implementation. Our empirical results show that using caching strategies that exploit the underlying process and workflow models significantly improve response times. With the tests it also became clear that caching strategies have different drawbacks, depending whether used in static or dynamic environments and respective static or dynamic access control constraints. Our hybrid caching approach combines strategies such that drawbacks from one strategy can be compensated by the strengths of another one, and vice versa.

While we only tested scenarios for caching access control decisions, we are convinced that this also holds for caching other types of information such as internal or external context information (e. g., the availability of resources or machines, or specific data objects from back-end systems, etc.) that are required for performing a specific task.

We see several lines of future research: First, different dynamic aspects of business applications need to be considered. For example, induced by changes in the distribution of process types being active (e. g., payroll processes are executed only once every month). Second, strategies that guarantee a bounded cache size, e. g., based on the least-frequently used principle, should be integrated into the different caching strategies. While this will decrease the overall performance, it prevents the unlimited growth of the cache which could occur in highly dynamic scenarios. Third, the most efficient caching strategies (PAC and SAAM) depend on the underlying access control model. Thus, it seems to be interesting to extend our comparison to further access control models (e. g., Bell-LaPadula [3]). While for SAAM there exists already a variant supporting Bell-LaPadula [6], the heuristics for PAC [14] need to be adapted.

References

[1] Basel II: International convergence of capital measurement and capital standards. `http://www.bis.org/publ/bcbsca.htm`, 2004.

[2] L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *Security and Privacy*, pages 81–95. IEEE Computer Society, 2005.

[3] D. E. Bell and L. J. LaPadula. Secure computer systems: A mathematical model, volume II. In *Journal of Computer Security 4*, pages 229–263, 1996.

[4] K. Borders, X. Zhao, and A. Prakash. CPOL: high-performance policy evaluation. In V. Atluri, C. Meadows, and A. Juels, editors, *Computer and Communications Security*, pages 147–157. ACM Press, 2005.

[5] H.-T. Chou and D. J. DeWitt. An evaluation of buffer management strategies for relational database systems. In *VLDB*, pages 127–141. VLDB Endowment, 1985.

[6] J. Crampton, W. Leung, and K. Beznosov. The secondary and approximate authorization model and its application to Bell-LaPadula policies. In *SACMAT*, pages 111–120. ACM Press, 2006.

[7] T. Curran, G. Keller, and A. Ladd. *SAP R/3 business blueprint: understanding the business process reference model*. Prentice-Hall, Inc., 1998.

[8] M. Evered and S. Bögeholz. A case study in access control requirements for a health information system. In *ACSW Frontiers*, pages 53–61. Australian Computer Society, Inc., 2004.

[9] J. Hu and A. C. Weaver. Dynamic, context-aware access control for distributed healthcare applications. In *Pervasive Security, Privacy and Trust*, 2004.

[10] V. Kapsalis, L. Hadellis, D. Karelis, and S. Koubias. A dynamic context-aware access control architecture for e-services. *Computers & Security*, 25(7):507–521, 2006.

[11] R. Karedla, J. S. Love, and B. G. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, 1994.

[12] G. Karjoth. Access control with IBM Tivoli access manager. *ACM Transactions on Information and System Security*, 6(2):232–257, 2003.

[13] M. Kohler and A. Schaad. Proactive access control for business process-driven environments. In *Annual Computer Security Applications Conference*, 2008.

[14] M. Kohler, A. D. Brucker, and A. Schaad. Proactive caching: Generating caching heuristics for business process environments. In *CSE*, pages 207–304. IEEE Computer Society, Aug. 2009.

[15] N. Megiddo and D. S. Modha. ARC: A self-tuning, low overhead replacement cache. In *USENIX FAST*, pages 115–130. USENIX Association, 2003.

[16] J. Nielsen. *Usability Engineering*. Academic Press, 1993.

[17] R. S. Sandhu, D. F. Ferraiolo, and D. R. Kuhn. The NIST model for role-based access control: towards a unified standard. In *RBAC*, pages 47–63. ACM Press, 2000.

[18] The JBoss Group. jBPM. `http://www.jboss.com/products/jbpm`, 2008.

[19] F. Turkmen and B. Crispo. Performance evaluation of XACML PDP implementations. In *SWS*, pages 37–44. ACM Press, 2008.

[20] Q. Wei, J. Crampton, K. Beznosov, and M. Ripeanu. Authorization recycling in RBAC systems. In *SACMAT*, pages 63–72. ACM Press, 2008.