Chapter 10

Experimental Results

There are three principal means of acquiring knowledge available to us:
observation of nature, reflection, and experimentation.
Observation collects facts; reflection combines them;
experimentation verifies the result of that combination.
Denis Diderot, On the Interpretation of Nature

We identified the flexible event composition and the system's performance as the two main challenges for adaptive integrating event notification services. In chapters 5 to 8, we introduced methods and algorithms that address these challenges. This chapter serves as a proof of concept for our approach towards an adaptive integrating ENS. The objectives of this chapter are threefold:

A Demonstration of the qualitative adaptivity of our A-MEDIAS system,

B Quantitative analysis of the value and attribute measures for primitive filter algorithms, and

C Experimental evaluation of the efficiency of our primitive and composite filter algorithms in A-MEDIAS. Additionally, we demonstrate the quantitative adaptivity of the system.

The chapter's organization follows these objectives: In Section 10.1, we present a case study for application-dependent event notification (qualitative adaptation) as proposed in chapters 5 and 8 for a facility management scenario. The case study shows the different profile evaluations for two clients with similar profiles depending on the event source and application scenario. The clients are notified about the various appropriate events without having to change their profiles according to the application characteristics.

In Section 10.2, we present the results of a simulation of our primitive–event filter algorithm introduced in Chapter 7. We simulated the effect of different selectivity measures under various profile and

event distributions. The effectiveness of our measures is demonstrated by major performance gains for typical situations in event notification services.

In Section 10.3, we present efficiency tests for the filter algorithms implemented in our A-MEDIAS system. We presented two filter algorithms in this thesis: a distribution-based enhanced tree-algorithm for the filtering of primitive events and our single-step algorithm for the filtering of composite events (cf. Chapter 7). In these efficiency tests, we compare the performance of our algorithms with the results of related filter algorithms. Additionally, we analyze the influence of parameters, such as profile distribution and the number of profiles, on the filter performance. Finally, we present a case study for a facility management scenario that shows the effect of adapting the filter algorithms to changing profile and event distributions. We present a detailed analysis of the results obtained during the experiments. In Section 10.4, we summarize the main results.

10.1 Application-Dependent Event Composition

In this section, we present a case study for a facility management application in which we show the effect of the parameterized event algebra and the qualitative adaptivity implemented in A-mediAS. Two clients with similar profiles are notified about different composite events depending on the event sources and the application context. In this case study, we show the effect of changing applications on an adaptable profile evaluation. Six different sources are covered: Four temperature sensors (T1 to T4) and two air conditioning systems (AC1 and AC2) are monitored. The sources are shown in Figure 10.1, left. The event sources do not change within our case study; therefore, we omit the source profile descriptions. The events sent by these sources are shown in the event histories in Figure 10.6(a), top.

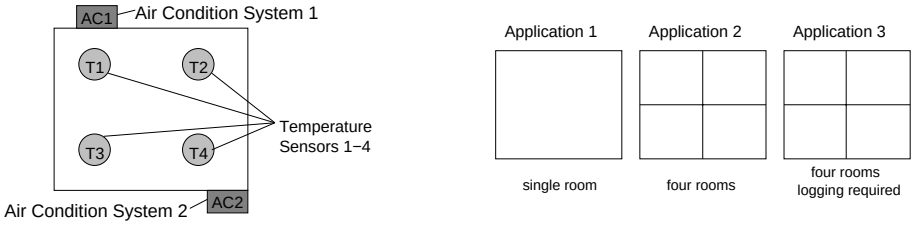


Figure 10.1: Case Study: Event sources and applications

We consider two technicians A and B that each define a profile to monitor the effects of air conditioning failures. Each of the technicians is responsible for one air conditioning system. Due to an insurance contract, after a system failure each of the two technicians has to inspect their affected rooms for at least one week. The profile definitions are given in Figure 10.2.

We consider three application variants that each describe a different usage form of the building (see Figure 10.1, right). An application expert defines the application profiles as shown after each application description.

In Application 1, the four temperature sensors refer to a single room in the building. Each technician is notified about the effects of the air conditioning system for which he or she is responsible. The technicians have to check the affected rooms for at least one week. The application profile is shown in Figure 10.3.

```
Profile Technician A:                    Profile Technician B
  COMP (                                   COMP (
    PRIM( air = 1, alarm = "failure" ),      PRIM( air = 2, alarm = "failure" )
      FIRST                                    FIRST
    BEFORE                                   BEFORE
    PRIM(temp = *, alarm = "hot"),           PRIM(temp = *, alarm = "hot"),
      ALL                                      ALL
    MULTI=ALL                                MULTI=ALL
    EVAL=FINALLY                             EVAL=FINALLY
    EVALTIME = 7 days )                      EVALTIME = 7 days )
```

Figure 10.2: Profiles for technicians A and B as used in the case study

```
APPLICATION ( ID = 1
  SOURCE (type = "air",
    EIS ((FIRST : FIRST), (LAST : LAST), (ALL : ALL)))
  SOURCE (type = "temperature",
    EIS ((FIRST : FIRST), (LAST : LAST), (ALL : FIRST))))
```

Figure 10.3: Application profile for Application 1

In Application 2, the single room has been divided by movable walls into four separate rooms. Each of the four temperature sensors refers to one of the rooms. Similarly to Application 1, the technicians have to check affected rooms for at least one week after a failure in the air conditioning system for which they are responsible. The application profile is shown in Figure 10.4.

```
APPLICATION ( ID = 2
  SOURCE (type = "air",
    EIS ((FIRST : FIRST), (LAST : LAST), (ALL : ALL)))
  SOURCE (type = "temperature",
    EIS ((FIRST : FIRST), (LAST : LAST), (ALL : ALL))))
```

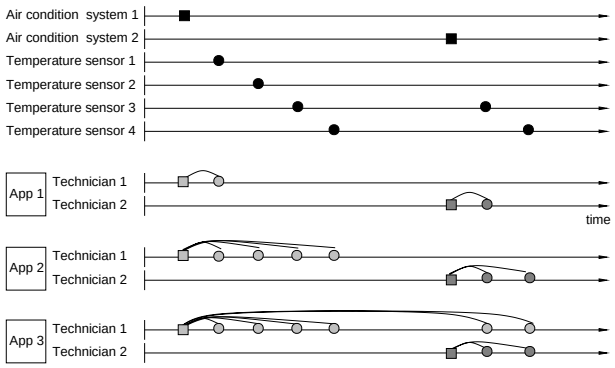
Figure 10.4: Application profile for Application 2

In Application 3, the situation in the building is as described for Application 2. The insurance contract has been changed: All temperature events after a system failure have to be monitored (logged) for at least one week. The application profile is shown in Figure 10.5.

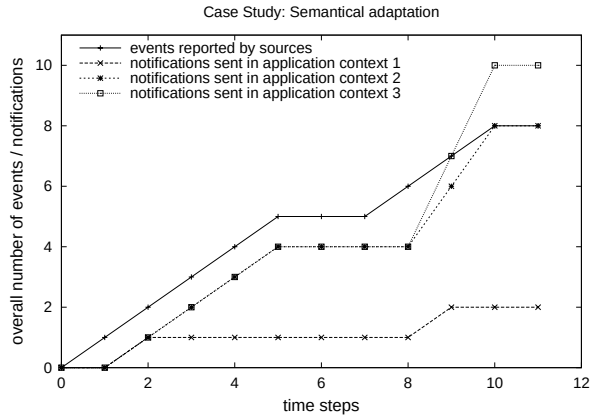
```
APPLICATION ( ID = 3
  SOURCE (type = "air",
    EIS ((FIRST : FIRST), (LAST : LAST), (ALL : ALL)))
  SOURCE (type = "temperature",
    EIS ((FIRST : FIRST), (LAST : LAST), (ALL : ALL)))
  EIC (logging))
```

Figure 10.5: Application profile for Application 3

After the translation according to the application profiles, the technicians' profiles are evaluated with new parameter settings. Figure 10.6(a) shows the composite events each technician is notified about in each of the three application scenarios. The arcs refer to the event pairs contributing to each composite event. The influence of the different application profiles is shown by their effect on the event



(a) Event sources and notifications for three scenarios, arcs refer to composite events



(b) Number of events and notifications

Figure 10.6: Case Study: Adaptation of event composition to application scenarios

composition (qualitative adaptation). Thus, the case study gives an example for the adaptive behavior of A-MEDIAS regarding the profile semantics.

Figure 10.6(b) shows the number of notifications that are sent by the A-MEDIAS system in each of the application scenarios. Thus, the qualitative adaptation additionally influences the system load (because matched events cause higher filter load than unmatched events, see Equation 7.1). This calls for the quantitative adaptation of the system, which is shown in the next two sections.

10.2 Efficiency Simulation of Primitive–Event Filtering

In this section, we present the results of a simulation of different selectivity measures for primitive–event filtering. The selectivity measures have been introduced for our tree-based filter algorithm for primitive events in Chapter 7 (cf. pages 104 and 105). We implemented and tested our enhanced tree algorithm (see Algorithm 7.2). The tree-reordering has been simulated according to the selectivity measures V1–V3 and A1–A2. Recall from Chapter 7 that V1–V3 describe reorderings of values in each attribute level

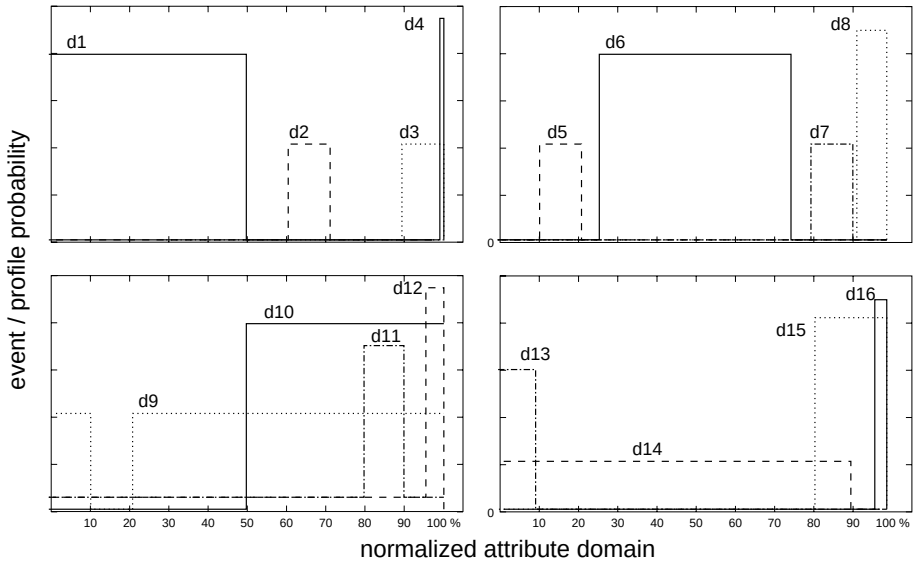


Figure 10.7: Example distributions used in the simulation of primitive filter algorithms, applied to a normalized attribute domain (0–100%)

and A1–A2 describe reorderings of the attribute-levels in the profile tree. For the presented simulation results, we set stronger focus on the value reordering because the attribute reordering is implemented and extensively tested in A-MEDIAS. We tested 60 variants of Gauss and uniform distribution with different parameter settings for the distributions of events and profiles (d1 – d60). We recognize the difference between distribution and parametrization of a distribution. However, for simplicity, we refer to the different distribution parameterizations as *distributions*. Figure 10.7 displays the selection of distributions that we refer to within the discussion of the simulation results. The graphs do not precisely describe each function, but give an impression of the distribution characteristics. Additionally, we used normally distributed data (gauss) and equally distributed data completely covering a domain D (equal).

Value Reordering. We tested each of the 60 distributions with 8 different orderings of attribute values within each tree-level plus binary search in four test scenarios. For brevity, we present selected test results for different combinations of distributions in events P_e and profiles P_p using the following test scenario:

Test-scenario for value selectivity (TV): Profiles and event message types with one attribute; empty zero-subdomain $D_0 = \emptyset$ for all attributes (full profile tree, all variations), set of all possible events. The mean number of filter steps is computed based a the combination of the number of operations in our simulation and the event distribution (according to Equation 7.3, Page 103).

Figure 10.8 displays simulation results for three search strategies in the values of an attribute-level of the profile tree: linear search in the alphanumerically ordered values (natural order search), linear search in values ordered with descending event distribution according to Measure V1 (event order search), and binary search. Results for various combinations of distributions are presented; the labels refer to the

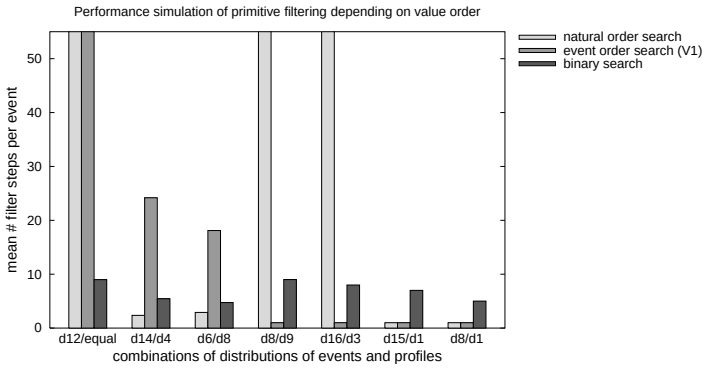


Figure 10.8: Simulation results for value reordering in profile tree (TV) showing the mean number of filter steps based on search strategy: linear search in values with natural order (natural order search) and with descending event distribution order (event order search), and binary search. Truncated bars represent test results that exceed the diagram's limit of 55 filter steps.

graphs in Figure 10.7. It can be seen that natural and event order have varying filter time, while binary search provides balanced results. The figure demonstrates that there is no "perfect" approach; depending on the distributions, different ordering strategies provide best performance. For certain applications, typical distributions of events and profiles exist. For these applications, specific filter approaches are appropriate. For example, the ordering according to event distribution shows best performance for distributions with *small peaks*: A small range of events is requested by many clients, while a wide range of events is not matched by the profiles, e.g., as in d8/d9 and d16/d3. These distributions are typical in event notification services, e.g, in a scenario for a catastrophe warning system.

Formally, linear search in values with descending event-distribution order (event order) is faster than binary search if $E(X) < \log_2(2p-1)$, assuming similar handling of non-matching events. This is a result from our analysis in Chapter 7. The selectivity based on the event order is a fragile measure, not robust to changes in the distributions. Reordering based on this measure is therefore recommended for systems with stable distributions.

Figure 10.9 displays the results of value reordering based on measures V1–V3 (for the definition of measures V1–V3 see Page 104). Again, for selected cases the reordering according to event distribution provides best performance. The profile-based reordering (V2) supports profiles to which many clients have subscribed – this reordering leads to a decreased average-case performance with respect to the events. The reordering based on Measure V3 follows a middle course: Frequent events that are of high interest for clients are supported, the average-case performance is lower than in the event-based order but events that have no subscribers are postponed. Both profile dependent measures support client groups with similar interest.

Figure 10.10 shows the mean filter steps per event (in Figure 10.10(a)), per profile (in Figure 10.10(b)), and with respect to events and clients (in Figure 10.10(c)). Note the differing scale in Figure 10.10(c). The tests have been performed using equally distributed events (equal), events with descending probability (falling) and equally distributed events with a small peak. Small peaks cover only a small fraction of the domain, such as distribution d16. The given number refers to the probability of the peak-values.

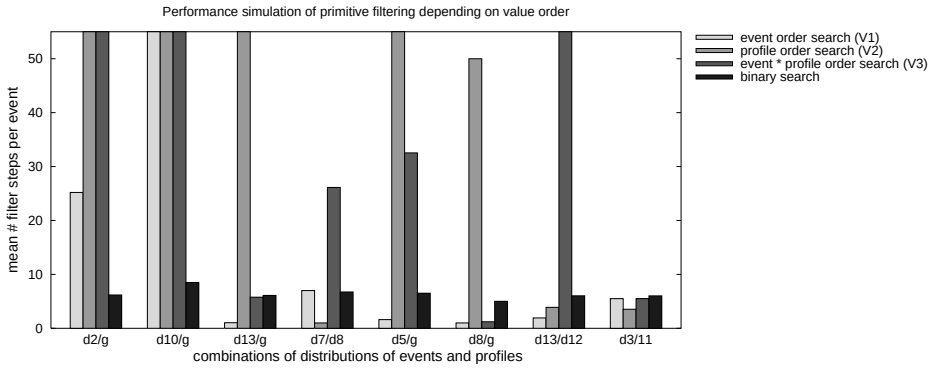


Figure 10.9: Simulation results for value reordering in profile tree (TV) showing the mean number of filter steps based on search strategy: linear search in values with (V1) descending event distribution order (event order search), (V2) descending profile distribution (profile order search), and (V3) descending event and profile distribution (event*profile order search), and binary search. Truncated bars represent test results that exceed the diagram's limit of 55 filter steps.

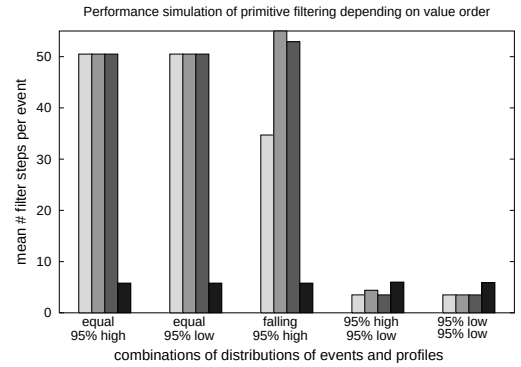
High and low refers to the location of the peak at the higher or lower values of the attribute domain.

We selected these distributions to discuss the influence of reordering according to the profile-dependent measures V2 and V3 on the performance. For equally distributed events, different profile distributions did not influence the mean number of filter steps per event, but the mean number of filter steps per profile (see data blocks 1 and 2). Here, the profile-dependent reordering (V2 and V3) significantly improves the performance per profile (block 2). The blocks 4 and 5 in Figure 10.10(b) and Figure 10.10(c) show that the mean number of filter steps per profiles and the mean number of filter steps per profiles and events may differ for situations with a minimal mean number of filter steps per event (block 4 and 5 in Figure 10.10(a)). Profiles with a high number of interested clients may have higher priority. The results presented here may be used to ensure best performance for high-priority profiles, i.e, by value ordering according V2 and V3.

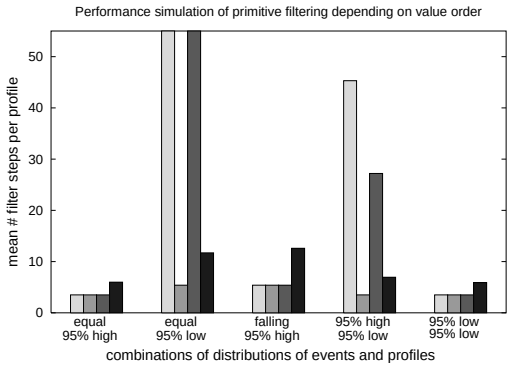
In summary, algorithms based on V2 and V3 lead to inferior mean filter time per event, but to faster notifications for profiles with high priority. For filter components operating in their optimal working point (i.e., $freq_{events} \ll freq_{filter}$) events do not queue. Thus, our algorithm improves the performance for selected profiles because fast filtered events are not slowed down by other events.

Attribute Reordering. For selected profile distributions, different attribute orderings A1 and A2 have been tested (definition of measures A1 and A2 see Page 105). We present selected test results for the following scenario:

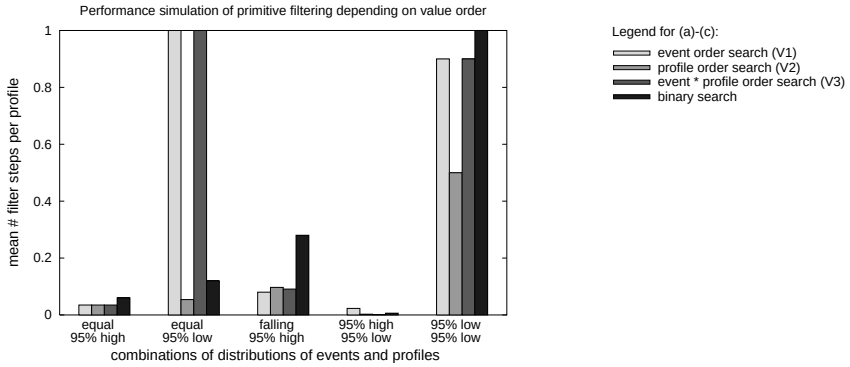
Test-scenario for attribute selectivity (TA): Profiles and event message types with five attributes; various sizes of zero-subdomains for the profile attributes (full profile tree, all variations), set of all possible events. The mean number of filter steps is computed based on the combination of number of operations in our simulation and the event distribution (according to Equation 7.4, Page 103). Assuming independence of the attributes, the tests have been performed using the overall distribution of events for each attribute, not conditional distributions.



(a) Mean number of filter steps per event



(b) Mean number of filter steps per profile

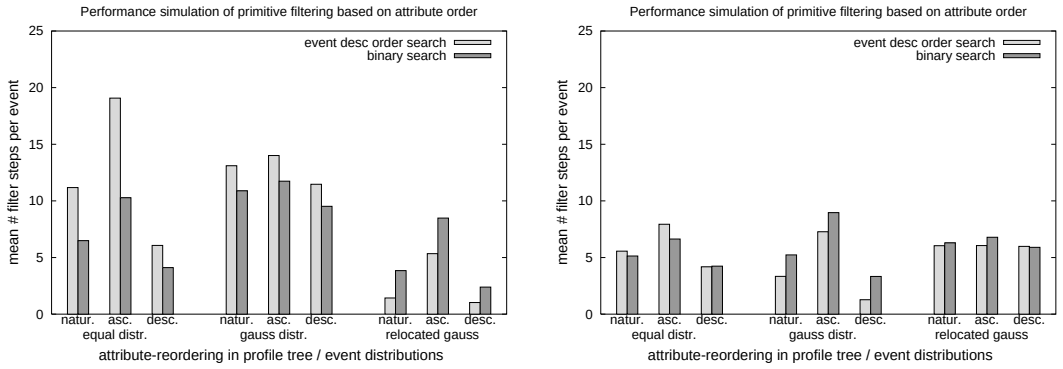


(c) Mean number of filter steps per event and profile

Figure 10.10: Simulation results for value reordering in profile tree (TV) showing the mean number of filter steps based on search strategy: linear search in values with (V1) descending event distribution order (event order search), (V2) descending profile distribution order (profile order search), and (V3) descending event and profile distribution order (event*profile order search) and binary search. Truncated bars represent test results that exceed the diagrams' limits of 55 and 1 filter step(s) respectively.

Here, we discuss the results of two test sets for the profile distributions. The first test set uses profiles whose attribute values have distributions with peaks of widths from 10 to 80 percent. That means that their respective zero-subdomains cover 90 to 20 percent. For example, d8 has a peak width of 10 percent and the zero-subdomain covers 90 percent (see Figure 10.7). The second test set uses profiles whose attribute values have distributions that vary only slightly. For both test sets of profiles, we applied three different event distributions: equal, Gauss, and relocated Gauss (center of the distribution shifted to the high values).

The test results are shown in Figure 10.11(a) and 10.11(b). The three tree-orderings refer to the natural order of the attributes, i.e., ordered according to the attributes' index-numbers (natur.), ascending order (asc.) and descending order (desc.) according to the attribute selectivity. Note that the ascending order describes the worst-case scenario, because the least selective predicates are set at the top of the profile tree.. We show the results of both value selectivity-based search (V2) and binary search. The data



(a) Wide differences in attribute distributions in profiles

(b) Small differences in attribute distributions in profiles

Figure 10.11: Simulation results for attribute reordering in profile tree (TA) showing the mean number of filter steps based on attribute order and search strategy. Attribute order in the profile tree: as defined in event type (natural), with ascending attribute selectivity (asc.), or with descending attribute selectivity (desc.)

for binary search vary for the various orderings, because non-matching events are detected differently. Note that the influence of the attribute selectivity measures is shown in the different blocks, whereas the additional effect of the value selectivity measure (i.e., the search strategy in the attribute values) is shown by different colors.

For equally distributed event data, the effect of Measure A1 equals that of Measure A2. Therefore, the desc-results in the left blocks in both figures 10.11(a) and 10.11(b) refer to A1 as well as A2. Recall from Chapter 7 that Measure A1 is only appropriate for equally distributed event data. The effects of Measure A2 for non-equally distributed event data are shown in the center and right blocks of both figures. Figure 10.11(a) (left) displays the data for equally distributed events. Figure 10.11(a) (center) displays the results of a similar test with Gauss-distributed events. In both cases, the descending event-based order according to Measure A2 (desc.) has a positive impact on the performance. However, the value reordering according to Measure V2 is not faster than the binary search since $E(X) > \log_2(2p-1)$.

Figure 10.11(a) (right) refers to a similar profile tree. We now consider a relocated Gauss-distribution that features especially high probability for events referring to zero-subdomains. Reordering of the tree shows best performance if ordered according to Measure A2 (desc.). Here, the value reordering according to Measure V1 is faster than binary search because a significant part of the events map onto the zero-subdomain and $E(X) < \log_2(2p-1)$. In all three test series, the reordering according Measure A2 (desc.) provided best performance.

Figure 10.11(b) displays the data for equally distributed events (left), for events distributed according to Gauss (center) and to a relocated Gauss (right). Similar patterns can be found, the ascending event-based reordering improves the filter performance. In this experiment, the relocated Gauss-distribution led to an overall lower performance.

We conclude from our analyses above and in Chapter 7 that attribute reordering according to measures A1–A3 can be used to reject unmatched events early: Reordering with Measure A1 is useful for

equally distributed data, measures A2 and A3 are appropriate for all event distributions. Measure A3 is costly to obtain and therefore only sensible for applications with stable distributions. In our A-MEDIAS system, we implemented measures A1 and A2. The efficiency tests for this implementation are presented in the next section.

10.3 Efficiency Tests of the A-MEDIAS Filter Engine

In this section, we evaluate the filter performance of the A-MEDIAS system. We examine the different filter algorithms and the influence of various workload parameters on the performance. First, the experimental environment is described. Then, we present performance results for profiles regarding (a) primitive events, (b) composite events, and (c) both primitive and composite events. From the variety of characteristics that are worth investigating, we focus on those that are crucial for adaptivity and integration: influences of distributions of events and profiles and related characteristics. We define the hypotheses on which our tests are based in the beginning of each part. Finally, we show the influence of filter algorithm adaptation on the overall performance of the A-MEDIAS filter engine in two case studies.

10.3.1 Experimental Environment

The A-MEDIAS system is implemented as a distributed system consisting of a network of servers. Unless stated differently, we evaluate the behavior of the stand-alone event notification server (see Figure 9.3) without concurrency or event bulk load facility. The experiments were performed on a personal computer (Win2000) with 1266 MHz and 512 MB main memory. The events and profiles were created using an event and profile generator for automatic creation of test cases. The tool randomly generates events and profiles based on predefined distributions of attribute values. In the experiments, each client profile contains one predicate on each of the attributes defined for the respective event type. We created different workloads by changing the workload parameters of the event and profile generator. A list of workload parameters is given in Table 10.1.

Parameter	Range	Description
N_p	5,000 – 55,000	Number of profiles
N_e	5,000 – 55,000	Number of events
N_a	1 – 5	Number of attributes
P_w	1% – 10%	Wildcards in profile predicates
P_p	PD0 – PD4	Profile distributions
P_e	ED0 – ED5	Event distributions

Table 10.1: Workload parameters for performance tests on filter algorithms

Additionally, we used various profile and event distributions. If not explicitly stated differently, a uniform distribution of attribute values is applied in events and profiles ($ED0$ and $PD0$ in Table 10.2). For the profile distributions, the profile attributes cover the given percentages (dense uniform distribution, starting at the lower end of the attribute domain) of each of the five attribute domains as shown in Table 10.2, left. These distribution forms are extensions to the distributions defined in Section 10.2.

P_p	Profile Distribution (in %)
$PD0$	$100/100/100/100/100$
$PD1$	$50/40/30/20/10$, (small steps)
$PD2$	$50/10/40/20/30$, (small steps)
$PD3$	$10/20/30/40/50$, (small steps)
$PD4$	$100/80/60/40/20$, (wide steps)

P_e	Event Distribution
$ED0$	100% uniformly covered
$ED1$	20% uncovered / 80% uniformly covered
$ED2$	50% uncovered / 50% uniformly covered
$ED3$	100% uniformly covered / 0% uncovered
$ED4$	50% uniformly covered / 50% uncovered
$ED5$	20% uniformly covered / 80% uncovered

Table 10.2: Distributions of profiles and events used in our performance tests of filter algorithms

For the event distributions, the attribute values cover the given percentages of each attribute domain. For a given test-set of events, all attributes have the same distribution. Each attribute value distribution holds the characteristics (dense uniform distribution, starting at the lower end of the attribute domain) as shown in Table 10.2, right.

We measured the performance of the system as the *filter time* to find all matching profiles for each of the incoming events. For the filter time per event, the mean value of the filter time for a number of events N_e is used. The costs of creating the events and profiles, as well as the profile representation are not included in the measured filter time.

10.3.2 Efficiency of Primitive–Event Filtering

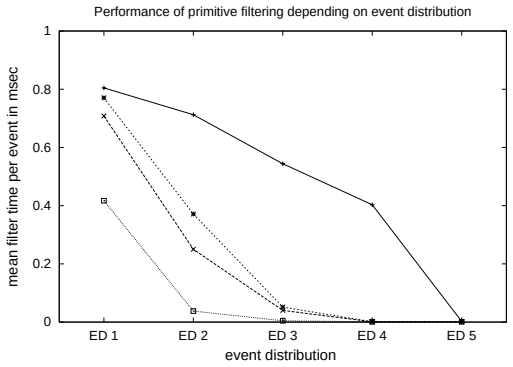
We now describe the results of three performance experiments and a case study to evaluate the performance of primitive–event filter algorithms. Within the tests, we varied the number of events, the number of profiles, and the fraction of wildcards in profile predicates.

Performance Experiments. The following list presents the hypotheses that our performance tests for primitive filters are based on.

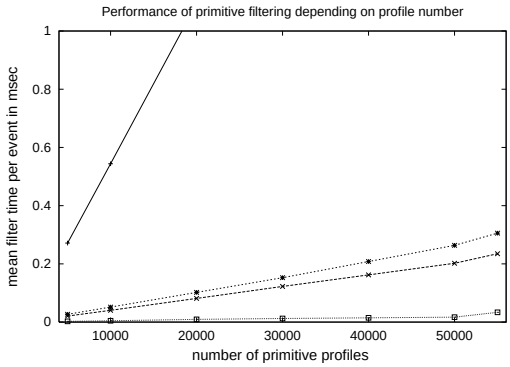
H1. Event and Profile Distribution: The filter time of the tree-algorithm decreases if distribution-based tree optimization is performed (using Measure A2 as implemented in A-mediAS). The larger the subdomains in the profile attributes for which no event is defined, the faster is the algorithm. The hypothesis is supported by the theoretical analysis in Chapter 7 and the simulation results in Section 10.2.

H2. Number of Profiles: Using a static collection of events and a stable profile distribution, the filter time increases with increasing number of profiles. The higher the coverage of the event space by the profile set (which depends on the profile distribution), the longer is the filtering time. This hypothesis is supported by the algebraic analysis of our basic performance model in Chapter 7 (see Equation 7.1, Page 95).

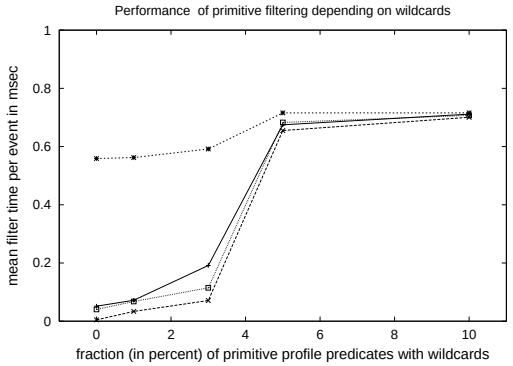
H3. Profile Similarity: The closer the similarity of the profiles, i.e, the more similar the profiles in the profile set, the lower is the influence of the tree optimization. The same effect has a higher coverage of the event space by the profile set (e.g., caused by don't-care edges in the tree). This hypothesis is supported by the definition of Measure A2: The zero-subdomains of the profile set are used to reject unmatched events early.

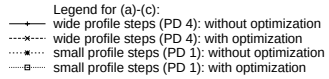


(a) Filter time based on event distribution, $N_p = 10,000$, $N_e = 50,000$, $N_a = 5$, $P_w = 0$, $P_p = PD1/PD4$, $P_p = ED1$–$ED5$



(b) Filter time based on number of profiles, $N_p = 1,000$–$50,000$, $N_e = 50,000$, $N_a = 5$, $P_w = 0$, $P_p = PD1/PD4$, $P_e = ED0$





(c) Filter time based on wildcards in profiles, $N_p = 10,000$, $N_e = 50,000$, $N_a = 5$, $P_w = 0 - 10\%$, $P_p = PD1/PD4$, $P_e = ED0$

Figure 10.12: Filter times for primitive events depending on (a) event distribution, (b) number of profiles, and (c) fraction of wildcards in profiles

We do not analyze the influence of the number of events. Because every event is filtered after the processing of the preceding one, the number of events does not influence the mean filter time per event. We show the test results for selected profile distributions P_p and event distributions P_e.

H1–H3: Influences on primitive filter time. Figure 10.12(a) shows the filter time under varying event distributions. Most events are matched in event distribution $ED1$, least in $ED5$. Figure 10.12(b) displays the results for the filter time under a varying number of profiles. The optimized tree shows a higher performance than the filtering in natural attribute order (the arbitrary order of the attributes in the event type). For example, the filter time for $0,000$ profiles with distribution PD4 using our optimized algorithm takes about $0.1msec$ per event while the original algorithm (with binary search) takes more than $1msec$. This is equivalent to a performance gain of 90 percent. Additionally, we see that our prototype system fulfills the performance requirements for a maintenance service as identified by analysis of our performance model (see Chapter 7 at Page 97).

Figure 10.12(c) shows the influence of wildcards in the profile queries. Wildcards are profile attributes that are not restricted but may take any value of the attribute domain. The higher the fraction of wildcards, the more similar are the performance results of the algorithms. The reason is that more events are matched if the profiles contain more wildcards. All four curves surge at 5% wildcards in the profiles. At this point, almost all events are matched by the profiles.

In summary, the results of the performance tests for the primitive filter algorithm in A-MEDIAS corroborate our hypotheses H1 to H3.

Case Study. We performed a case study that analyzes and illustrates the system's behavior under changing numbers of events and profiles, using only primitive profiles. Figure 10.13 shows the filter time for the following case: A system starts with $10,000$ profiles; $50,000$ events are filtered. Then, $10,000$ more client profiles are defined and another $50,000$ events are filtered. At last, $10,000$ additional profiles are defined, followed by $50,000$ events to be filtered. The profile sets are defined in the following order: $P_p^1 = PD1$, $P_p^2 = PD2$, and $P_p^3 = PD3$. Every possible optimization decision is shown: After each added profile set, the profile tree may be reordered or not. The legend refers to the three steps as listed, describing from left to right the steps undertaken in the particular approach, e.g., the label 'optimize / add / add' refers to a tree-reordering after the initial profile set followed by two added profile sets without tree reordering.

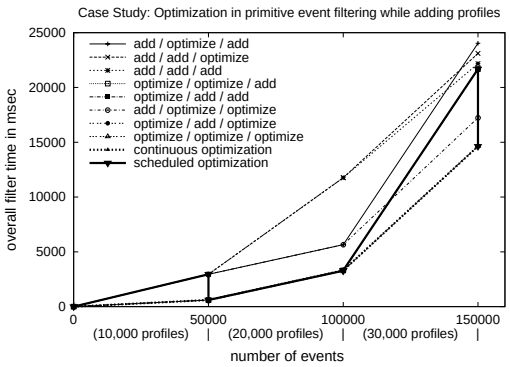


Figure 10.13: Case Study: Adaptation of primitive–event filtering to variable profile sets when using profile distributions $PD1$, $PD2$, and $PD3$; $50,000$ events after each $10,000$ profiles

The filter behavior under optimization is shown as thick line. Two modes may be used for adaptation: continuous analysis or scheduled analysis. For continuous analysis, the required tree-reordering may be applied to an off-line filter pool which is then exchanged with the running system. Using scheduled analysis, the profile tree is optimized based on a scheduled analysis of the profile set and the event history.

We see that under unfavorable conditions, an optimized tree with additional profiles has lower performance than a non-optimized tree. This results from profile distributions that require opposite optimizations: The optimized tree for one profile set is the worst-case constellation for another one. The approaches with regularly performed tree-optimization show best performance results. The results of the case study emphasize the importance of tree-optimization.

Our approach can be used to reduce the workload in resource critical environments, for example in mobile computing. Unnecessary event information is rejected as early as possible and the filtering can be optimized according to the application and client needs.

10.3.3 Efficiency of Composite–Event Filtering

In this section, the performances of different filter methods for composite events are evaluated: Our single-step method for filtering of composite events is compared to a two-step implementation. Performance tests of composite events underlie several parameters, e.g., the composite operator, the operator parameters, the time between the contributing primitive events of the composite event (composition distance), and the number of events and profiles. The following list presents the hypotheses that our performance tests for composite filters are based on.

H 4. Event distribution – event-composition order: The filter time for composite–event filtering increases when the event intervals between the contributing events decrease. This hypothesis is supported by the analysis of composite filter algorithms in Chapter 7: The tree-based algorithm (for comparison reasons implemented in A-MEDIAS) tests all potential composition partners while our single-step algorithm tests only event partners that actually occurred.

H 5. Number of profiles: Using a constant collection of events and profiles, the filter time increases with increasing number of profiles. The filter time depends linearly on the number of matching profiles. Unmatched profiles have less influence.

H 6. Profile distribution: The distribution of composite profiles describes the number of composite profiles that refer to identical primitive profiles. The cardinality κ (see below) strongly influences the single-step algorithm because overlapping primitive profiles influence the tree shape and, thus, the maintenance costs for the tree. Additionally, the profile reference lists of primitive profiles are influenced.

The evaluate performance, we measure both the filter time for primitive and composite events and the composition time per composite event. The filter time of composite events is the time between the occurrence of the last contributing event and the time the client notification is sent. The composition time is the time for the determination of the composition after the detection of the last contributing event. We additionally measure the overall filter time.

For the composition of events, several combinations of parameters and operators have been tested. For brevity, we restrict the presentation to profiles regarding simple event sequences of two events without temporal restrictions. The events are posted to the filtering mechanism as a continuous stream, such that the additional delay due to event frequencies is not considered here. We used events and profiles with one integer attribute ($N_a = 1$) and the attribute domain $[-100,000; 100,000]$. To eliminate the influence of profile overlap, only distinct profiles have been defined ($P_w = 0$). Additional workload parameters are shown in Table 10.3.

Parameter	Range	Description
g	1 - 1000	Number of event groups
N_p^c	5,000 - 55,000	Number of composite profiles
matches	first or all matches	Number of matches before profile removal
κ	(N to 1) or (1 to N)	Cardinality in $(E_1; E_2)$: number of E_1 profiles on number of E_2 profiles

Table 10.3: Additional workload parameters for composite-event filtering

For both, the two-step and the single-step method, we tested the following parameter settings:

1. The first event instance within a group of duplicate events is selected, subsequent instances of the same event class are ignored. If no duplicate event instances occur, as in most of our tests, the results for selecting the first event instance equal those for all instances and for the last event.

2. For the event composition, two opposite versions have been tested: single unique pairs (first match) and all pairs (all matches). These two variants model opposite filter behavior: For single unique pairs, the first occurring pair triggers a notification, afterwards the profile is removed. For all pairs, every possible combination of event instances triggers a notification.

3. We adopt the concept of cardinality κ as used in database design [GMUW02]. Two extreme types of sequences are used: In a set of n composite profiles, either (a) all second primitive profiles refer to the same first profile (1 to N), or (b) all first profiles refer to the same second profile (N to 1).

H4: Filter time depending on composition order. First, we evaluate the influence of the composition order on the filter time. We show the test results for 10,000 profiles regarding sequences $E_1; E_2$. We formed groups of E_1 events with an E_2 event at the end of each of the groups. The number of groups is referred to as g. Within each group, first come $10,000/g$ E_1 events and then one E_2 event that closes the group. For example, for $g = 1000$, every ten events of E_1 are followed by one event of E_2. For our tests, we varied the number of groups g. Figure 10.14 shows the influence of this event grouping on the filter time. It can be seen that the overall filter time for each event directly depends on the number of groups, i.e., on the number of interleaving E_2-events. Additionally, our single-step method needs only about

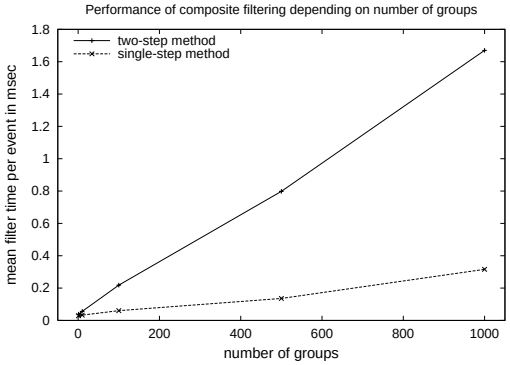


Figure 10.14: Filter times for composite events depending on composition order (event grouping), $g =$ 1–1,000, $N_p^c = 10,000$, $N_e = 10,000$, $matches = first/all$, $\kappa = (N \text{ to } 1)$

one eighth of the filter time required for the two-step method. This is equivalent to a performance gain of about 85 percent. The reason for this behavior is the algorithm's handling of matched contributing primitive events: In the two-step method, each E_2 profile carries a list of matching E_1-profiles. The complete list has to be checked after each E_2-event. In the single-step method, the E_2 references are only inserted after the occurrence of an E_1 event. Additionally, the E_2 do not carry a list of E_1-profiles; they directly trigger the notifications for those E_1-profiles that match events that already occurred.

H5: Filter time depending on number of profiles. We evaluated the influence of the number of profiles on the filter time. We show the results for sequences in a single group of events (see Figure 10.15(a)) and for sequences in 1,000 groups of events (see Figure10.15(b)).

Figure 10.15(a) and 10.15(b) show the filter time per event using the single-step and two-step methods for two different parameter settings: first match and all matches. For events in a single group (Figure 10.15(a)), both methods using all matches are faster than the methods using only the first matches. For the first matches, the profiles are removed after the matches, which causes maintenance costs.

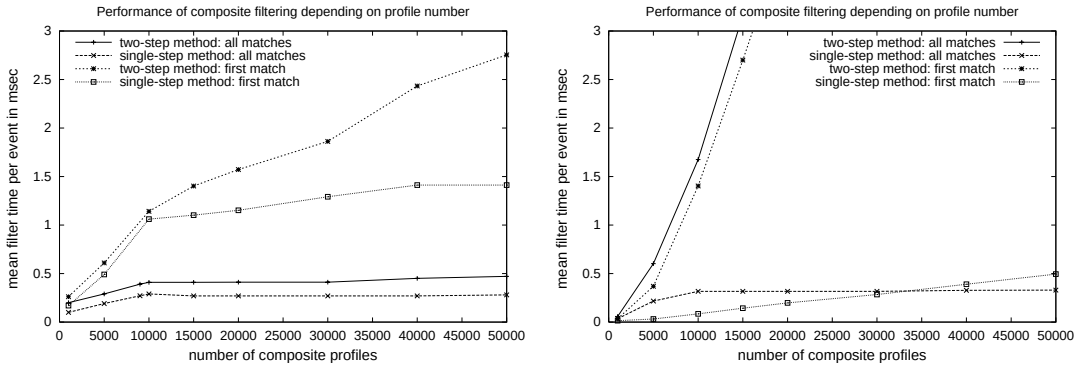
The single-step method for the first match is faster than the two-step method for the first match. In the first method, the profiles regarding the E_1-events are removed directly after the E_2-events have been detected. In the two-step method, all profiles are only removed at the end of the event detection, i.e., after the E_2-event. Therefore, a performance benefit due to a smaller profile pool applies to the single-step method, but does not affect the two-step method.

The single-step method for all matches is faster than the two-step method for all matches. For the first method, less profiles are stored in the profile pool, i.e., less profiles have to be evaluated. Additionally, the number of composite profiles to be checked in each step (in the composite pool or as auxiliary profiles) is lower for single-step.

In all four curves, the filter time increases linearly depending on the number of composite profiles between 0 and 10,000 profiles. In that interval, each of the profiles is matched by exactly one event pair. For more than 10,000 profiles, an increasing number of profiles remains unmatched. For the methods for all matches, a flattened logarithmical increase can be measured – the influence of the binary search on the profile values. For the methods with removal (single match), the removal costs have a major influence on the filter time.

In Figure 10.15(b), the effects already observed in Figure 10.15(a) are overlayed with the influence of the event grouping as shown in Figure 10.14. The decelerating influence of interleaving events (grouping) is strongest in the two-step method: Both versions of the two-step method suffer costs for the evaluation of the lists with potentially matching composition partners. The removal of matched profiles has a minor accelerating influence (due to reduced number of profiles) compared to the grouping costs.

The influence of the grouping on the single-step method is twofold – most significantly on the version with removal (first match). Compared to the corresponding filter time for one group (Figure 10.15(a)), the curve is lower but rises in a similar form. The shorter filter time is based on the lower initial filter time for 10,000 profiles, which is due to the shorter matching lists and an accelerating influence of the reduced number of profiles. The two curves cross at 30,000 profiles: The more profiles, the stronger is the influence of the profile removal. The exact crossing point of the curves depends on the efficiency of the implementation of the profile removal.



(a) Events provided in 1 event-group, $g = 1$, $N_e = 10,001$, $N_p^c = 10,000$, $matches = first/all$, $\kappa = (N \text{ to } 1)$

(b) Events provided in 1,000 event-groups, $g = 1,000$, $N_e = 11,000$, $N_p^c = 10,000$, $matches = first/all$, $\kappa = (N \text{ to } 1)$

Figure 10.15: Filter times for composite events depending on the number of profiles

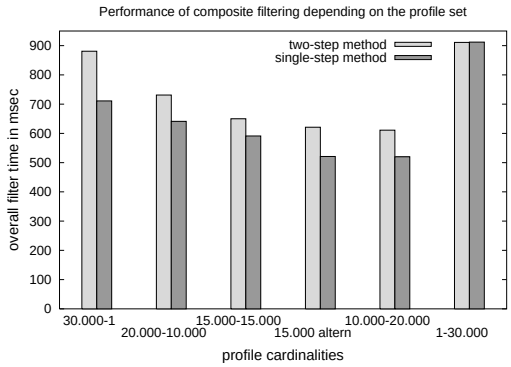
H6: Filter time depending on profile cardinality. The profile cardinality κ for a set of sequence-profiles $(E_1; E_2)$ describes the number of distinct profiles for E_1 and E_2. The cardinality is shown at the x-axis of the diagrams in Figures 10.16(a) and 10.16(b). In Figure 10.16 and 10.16(b), the overall filter times and composition times for different profile cardinalities are shown, respectively.

Figure 10.16(a) shows the overall filter time for 30,000 events, for the single-step and two-step methods. The filter time of the single-step method is in most cases less than the filter time for the two-step method. The filter time is equal only for the extreme case of one event starting off 30,000 sequences (1–30,000). The reason is the overhead for the tree-insertions for 30,000 profiles.

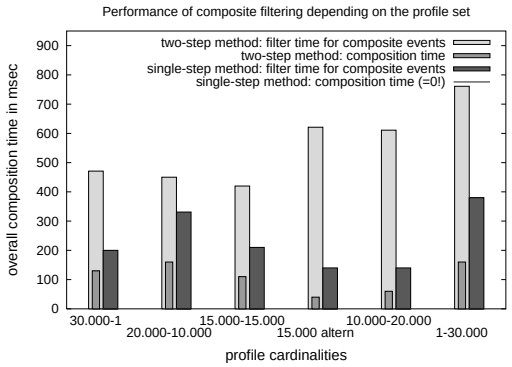
Figure 10.16(b) shows the composition times for the same experiments. Two values are shown for each test: the overall time needed for composing events and the additional time required after the last contributing event. As argued earlier, the latter defines, together with the primitive filter time for the last event, the filter time of composite events. As composition time for the single-step approach, we measured the time for the handling of auxiliary profiles. The composition time for the single-step method is shorter than that for the two-step method. The additional filter time for the composite events due to a composition step is zero. Moreover, the performance gain in the filter time is higher than the reduction of composition time. The reason for this behavior is the smaller profile tree.

We argued that for the efficiency analysis of composite-event filtering only the filter time of the last contributing event has to be considered. We have shown in our analysis that the single-step method efficiently minimizes this filter time for composite events to zero. Additionally, the overhead of unnecessary profile filtering is reduced and, therefore, the overall performance increases.

Filter time depending on algorithm implementation. For the implementation of our single-step method as introduced in Chapter 7, we evaluated two variations. We measured an influence of the data structures and methods used for implementation. As expected, the tree-maintenance methods have major influence on the performance since the single-step method is based on profile insertions and deletions. The results presented here have been measured for a direct implementation that removes and



(a) Overall filter time, $g = 1$, $N_p^c = 30,001$, $N_e = 30,000$, , $matches = all$, $\kappa = (N \text{ to } 1)$–$(1 \text{ to } N)$



(b) Overall composition time, $g = 1$, $N_p^c = 30,001$, $N_e = 30,000$, , $matches = all$, $\kappa = (N \text{ to } 1)$–$(1 \text{ to } N)$

Figure 10.16: Filter times and composition times for composite events depending on profile cardinality

adds profiles to and from the tree. Variations are the marking of profiles: Partial and auxiliary profiles are inserted into the profile pool at once instead of inserting and removing them from the pool. The profiles are marked if permitted for use and unmarked if the profiles are not valid.

Our implementation of the second variant results in an additional reduction of the filter time of up to 50 percent compared to the test results shown here. Details of the implementation and additional test results can be found in [Koe03]. The second approach is especially favorable for profile sets with large overlaps. Then, most profiles reference several clients; inserting or deleting profiles results in changes to those references.

Our hypotheses H4 to H6 have been confirmed by the test results for composite filter performance. In addition, the implementation structures have strong influence on the system's performance. The usability of the single-step method depends on the efficiency of the implementation of the algorithm and the used data structures. For further details see [Bit03, Koe03].

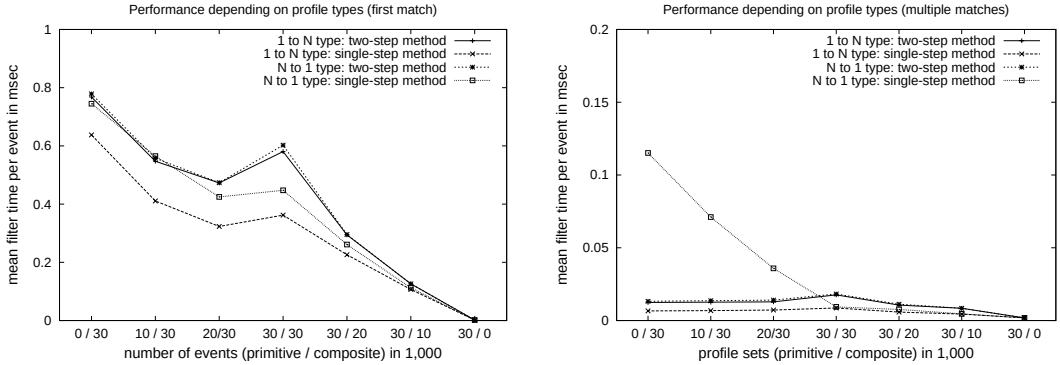
10.3.4 Efficiency of Primitive and Composite–Event Filtering

In this subsection, we present the results of experiments and a case study to evaluate the influence of the content of the profile set (i.e., the influence of the proportion of composite and primitive profiles in the set) on the filter performance.

Performance Experiments. We formulate the following hypothesis for our performance tests for primitive and composite filters.

H 7. Distribution of profiles and events: The ratio of primitive and composite profiles and their similarity influences the filter time. High similarity between primitive profiles and the contributing events of composite profiles lead to less tree reordering. Thus, the higher the overlap the shorter the filter time. In general, filtering of primitive profiles is faster than filtering of composite profiles.

The Figures 10.17(a) and 10.17(b) display the filter times per event under changing profile sets. Starting with 30,000 composite profiles, we successively added primitive profiles. In a second phase, we successively removed composite profiles until 30,000 primitive profiles were reached. Note that the number of profiles is changing in the profile sets. We show the results for both, (1 to N) and (N to 1) types of profiles (cf. Table 10.3).



(a) Filter time based on profile set (single matches), $g = 1$, $N_e = 50,000$, $N_a = 1$, $P_w = 0$, $P_p = PD0$, $P_e = ED0$, , $matches = first$, $\kappa = (N \text{ to } 1)/(1 \text{ to } N)$

(b) Filter time based on profile set (multiple matches), $g = 1$, $N_e = 50,000$, $N_a = 1$, $P_w = 0$, $P_p = PD0$, $P_e = ED0$, , $matches = all$, $\kappa = (N \text{ to } 1)/(1 \text{ to } N)$

Figure 10.17: Filter times for primitive and composite events depending on the profile set

For single matches as shown in Figure 10.17(a), the single-step method outperforms the two-step method in all tests. For multiple matches as shown in Figure 10.17(b)[1] on the (1 to N)-type composite profiles, the single-step method is slower than the two-step method if the number of composite profiles exceeds the number of primitive profiles (i.e., in the left part of the figure). The reason for this behavior is that under these conditions the profiles do not overlap. The more profiles are newly inserted into the tree, the slower the filtering. For profile sets with high overlap, the single-step method shows better performance than the two-step method. Consequently, the two-step method should be used for profile sets with only (1 to N) type profiles without overlap. Adaptation of the used profile strategy is necessary. The test results support our Hypothesis H 7 and additionally emphasize the importance of adaptability.

Case Study. We performed a case study that shows the system's behavior under changing numbers of events and profiles using primitive and composite profiles. Figure 10.18 shows the results of our case study regarding composite events. In the case study, we simulated the system behavior under the aspect of changing client profiles using a blend of different profile types: 10,000 profiles of (N to 1) type are inserted, then 10,000 profiles of (1 to N) type, and, finally, 10,000 primitive profiles. After each update of the profile set, 50,000 events are submitted to the system.

All possible optimization steps are shown (single-step vs. two-step filtering method). The methods for composite profile filtering applied at one time do not influence the performance at a later time. This behavior differs from the optimization used for primitive profile filtering (tree reordering), where earlier optimization steps influence later filter behavior. The reason is that not the complete profile pool

[1] Note the modified scale of the y-axis compared to single match tests.

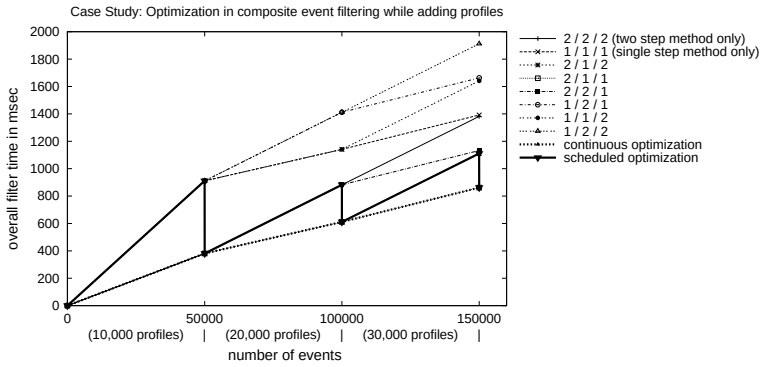


Figure 10.18: Case Study: Adaptation of composite–event filtering to variable profile sets when using profile cardinalities $(N$ to $1)$, $(1$ to $N)$, and primitive profiles; $50,000$ events after each $10,000$ profiles

structure is changed but only those parts used in the particular filter methods. The performance of our adaptive A-MEDIAS system is shown as thick line. Again, two modes may be used: the continuous analysis of the profile set with immediate adaptation or analysis and adaptation after n profiles have been changed. Both versions are shown in the figure. The results of the case study emphasize the importance of adaptive profile filtering for primitive as well as for composite events.

10.4 Summary

In this chapter, we presented the results of three analyses: (A) a case study of qualitative adaptivity, (B) a simulation of the selectivity measures for our enhanced filter algorithm fo primitive events, and (C) performance experiments of the filter algorithms implemented in A-MEDIAS. The A-MEDIAS system adapts to changing application context and new sources by translating the client queries according to an application profile. In a similar way, accuracy tolerances can be controlled. We have demonstrated this adaptive system behavior in a case study.

We have shown the effectiveness of our selectivity measures introduced for the primitive filter algorithm. Our distribution-based approach improves the average-case performance of tree-based algorithms. Our enhanced tree algorithm significantly reduces the filter time for primitive events for typical event and profile distributions in an event notification service. In extensive performance tests, we have shown that our single-step method significantly reduces the filter times for composite events with the effect that a composite event is detected as fast as a primitive event. Additionally, we have demonstrated that the single-step method improves the overall filter performance.

The system adapts to changing system load by selecting the optimal filter algorithms for both primitive and composite events. We evaluated the system performance under changing workload. The results of our case studies emphasize the importance of adaptive profile filtering for primitive as well as for composite events. Continuous adaptation of primitive and composite filter algorithms would cause an additional overhead for the filter engine. Therefore, the optimization frequency for a given system has to be determined under consideration of the tradeoff between optimization overhead and performance gain.