Leasing Variants in Distributed Systems

Michael Schneider  Markus Aleksy  Martin Schader
University of Mannheim
Department of Information Systems
schneider|aleksy|schader@wifo3.uni-mannheim.de

Makoto Takizawa
Tokyo Denki University
Department of Computers and System Engineering
taki@takilab.k.dendai.ac.jp

Abstract

In recent years, the leasing concept has become increasingly popular in the field of distributed systems; main examples are JINI and the introduction of leasing to the CORBA context. Nevertheless, no detailed analysis of leasing variants and their fields of application has been done yet.

In this paper, we give a systematic classification of possible leasing variants, discuss their advantages and disadvantages and point out their possible uses. In this context, we dispose of the restriction that resource claimants are not capable of waiting for a resource; an assumption made in hitherto literature for simplicity reasons. Furthermore, we consider possible mechanisms on which to base lease renewal decisions that go beyond the simple rules examined in previous works. In the process, we hint at how existing leasing approaches have to be enhanced to accomplish these relaxations.

1. Introduction

The leasing concept plays an important role in the architecture of Jini [5] and is supported by a detailed specification issued by Sun Microsystems [4]. Moreover, Aleksy et al. introduced the leasing concept to the CORBA environment by developing a CORBA leasing service [1, 2].

The basic idea of leasing is to govern access to resources by issuing leases for a predefined period of time instead of letting clients access resources unrestrictedly. A real-world leasing example are software licenses, which can be regarded as a lease between the software and the user. The potential benefits of leasing, which are in no way limited to the Jini or CORBA context, are described in the work of Jain and Kircher [3]. Among them are simplicity of resource management, efficient resource usage, and enhanced system reliability. These advantages are achieved by freeing the resource user from the responsibility of releasing resources explicitly. Leasing ensures that unused resources are released automatically and thus do not unnecessarily increase the system load. Another benefit of leasing is the possibility to implement a higher degree of fairness in distributed systems. This is illustrated by the following simple example: a service S provides clients with random numbers. The number of random numbers requested by client A is very high, whereas the number requested by client B is rather small. Without leasing, client B might have to wait a long time if client A's request is processed first. However, if leasing is used, client A can only occupy the service for the predefined leasing period and afterwards client B is served.

Since the foundations, motivations, benefits and examples of the leasing concept have been detailed in the cited literature, we give only a short introduction to leasing basics here. Generally, the concept of leasing involves four components. The basic tasks of these leasing components are presented below. The nomenclature is adopted from [2] since we plan to use their leasing service approach as a basis for an enhanced leasing service in future works.

Resource A resource provides application functionality that can be used by different clients. This broad definition increases the possible fields of application. For example, the provided functionality could be a service or database session. From the leasing viewpoint, the kind of functionality offered is considered opaque.

Claimant A claimant is a component that wishes to use a resource. In order to be able to do so, the claimant...
has to obtain a lease for the desired resource from the lessor component.

**Lease** A lease is the connecting link between a resource and a claimant. It represents permission for a claimant to use a resource for a specified duration. Moreover, a lease provides the functionality for releasing a resource prematurely. If the leasing variant allows for prolongation of the leasing time, i.e. lease renewals are possible, the lease also offers this functionality.

**Lessor** The lessor component is responsible for managing the association of resources with claimants by granting or denying leases. Moreover, the lessor component can be involved in the decision whether a lease renewal is granted or not.

**2. Motivation**

Recent leasing approaches like e.g. [1, 2, 5] are designed in a way that does not allow claimants to wait for a resource, i.e. claimants are either granted a lease or renewal request, or an exception is thrown indicating that the lease or renewal cannot be granted. This assumption seems rather restrictive, especially for claimants that depend on results from a certain resource. They might have to request a lease repeatedly and catch the thrown exceptions until they are granted the lease. This same logic has to be implemented in all claimants.

A more sophisticated solution would be to allow the claimant to wait for a resource up to a certain time limit that the claimant has to decide on itself. This could be done by offering methods for obtaining or renewing a lease that take the upper limit for the waiting time as parameter. The additional tasks created by this enhancement, like e.g. managing the waiting queues for the resources and the granting of leases to waiting resources, would have to be carried out by the lessor component. When discussing leasing variants in the remainder of this work, we will consider variants that allow claimants to wait for resources.

Although leasing variants are discussed in the existing literature (see [1, 2, 3]), this is done on a rather superficial level. Only simple cases like e.g. a renewal request is always granted, a renewal request is never granted, or a renewal request is granted depending on the total lease time of a claimant, are illuminated. More sophisticated mechanisms for making renewal decisions are only hinted at and even the effects of setting the parameters of the simpler leasing variants, like for example the upper time limit for the total lease time, are not discussed. In this work, we provide a systematic classification and in-depth analysis of leasing variants. Thereby, special focus is set on the possible mechanisms for deciding whether a lease renewal is granted and the effects of varying the parameters of these renewal decision mechanisms are described in detail. In particular, we examine renewal decision mechanisms that cannot be executed by the lease component alone but require communication between the lease and lessor component.

To this end, the following section specifies the components of a leasing variant. Subsequently, we conduct our analysis of leasing variants in Section 4.

**3. Components of Leasing Variants**

A leasing variant consists of the following components:

1. the **request processing policy**,  
2. the **scheduling algorithm** used to manage the possibly existent waiting queue for the resource,  
3. the **applied lease renewal policy**, and  
4. the **renewal decision rule**.

A leasing variant is always defined for a single resource, i.e. it describes the leasing process for a specific resource. Hence, different leasing variants can be used for the leasable resources in a system. The following sections detail the components of a leasing variant.

**3.1. Request Processing Policy**

We differentiate two request processing policies: *queue* and *no queue*. A *no queue* policy implies that it is not possible for a claimant to wait for the resource, i.e. the lessor component grants or denies a lease request based on whether the required resource is already occupied or not. If a claimant is denied a lease, it has to reapply.

A *queue* policy means that it is possible for claimants to wait for the resource. As mentioned above, the lessor component has to keep a waiting queue associated with the specific resource.

**3.2. Scheduling Algorithm**

If a queue policy is applied, a scheduling algorithm is used to manage the waiting queue associated with the resource in order to decide which of the waiting applications is next granted the lease. Scheduling algorithms are a well-researched field due to their importance in multitasking and multiprocessor operating systems. Therefore, advantages and disadvantages as well as efficient implementations are known for all scheduling algorithms. For a detailed description of the most relevant scheduling algorithms, we refer the reader to [6].

The following short description of the scheduling algorithms relevant to this work is adopted to leasing systems,
i.e. operating system scheduling concepts, like e.g. preemption, are of no importance in the leasing context and therefore not mentioned. The following scheduling algorithms are utilized in our leasing variants:

**First Come First Serve (FCFS) Scheduling** The waiting queue is realized as a classic First-In First-Out (FIFO) queue. A new application interested in a lease for the resource is added at the tail of queue. If the resource becomes available, the application at the head of the queue is granted the lease.

**Priority Scheduling** Priority scheduling assigns a priority to each waiting application. The waiting queue is realized as a priority queue. Again, if the resource becomes available, the application at the head of the queue, i.e. the application with the highest priority is granted the lease. If two applications have the same priority, FCFS is utilized. A general problem of priority scheduling algorithms is starvation, i.e. a situation in which the scheduling algorithm does not schedule a waiting application with a low priority for a very long time (see [6]). For example, this may happen in case of strongly utilized resources, where higher-priority applications are repeatedly added to the waiting queue. One way to solve this problem is to continuously increase the priority of applications after they have waited a certain time. This concept is called aging (see [6]).

An example of a priority scheduling algorithm is shortest remaining time (SRT) scheduling (see [6]). Each application is assigned the priority $1/t_l$ before entry into the priority queue, where $t_l$ denotes the estimated absolute lease time of the application. One example of how to incorporate the aging concept is to compute the priority as follows:

$$Priority = \frac{t_l + t_w}{t_l}$$

where $t_w$ denotes the waiting time of the application.

### 3.3. Lease Renewal Policy

The lease renewal policy determines whether it is possible for a claimant to renew the lease it presently holds for a certain resource. We identify two renewal policies, no renewal and renewal:

**No renewal** Leasing with a no renewal policy, also called strict leasing (see [2]), implies that the resource is withdrawn from its holder immediately after the lease expires, i.e. no renewal of the held lease is possible. Obviously, renewal decision rules are not required for this policy.

The general advantages of a no renewal policy are:

- A no renewal policy simplifies the resource management since no renewal decisions have to be made.
- The time a resource can be held by an application without actual need is restricted to one leasing period per leasing request.

The disadvantages of such a policy are:

- The claimant might need to subdivide its tasks if it is not granted the amount of time necessary to perform the entire computation. Partitioning of computation tasks can become quite complex and in some cases impossible.
- A no renewal policy increases the probability that a lease is unnecessarily withdrawn from its holder. This occurs if no other claimant is interested in the held resource, e.g. if the waiting queue is empty in case of a queue policy. Needless lease withdrawal leads to overhead caused by clean-up tasks on lease end and the process of reobtaining the lease.

Application of a no renewal policy is recommendable for time-critical applications. Moreover, a no renewal policy might be a reasonable choice for applications that fulfill one or more of the following criteria:

- resources are strongly utilized
- high value is set on fairness
- the majority of tasks are subdividable without problems
- the majority of tasks can be performed in a single lease period

**Renewal** Leasing with a renewal policy means that it is possible for the current lease holder to renew the lease for a certain period of time. To this end, the lease component offers a method for the claimant to renew the lease or a lease is created with an option for automatic renewal, i.e. the lease automatically tries to renew itself. The general advantages of leasing with a renewal policy are:

- Enhanced flexibility since subdividing of the client’s tasks can be avoided if a sufficient number of renewals is granted.
- Possibility to reduce overhead by avoiding unnecessary lease alternations.

The disadvantages are:

- The complexity of resource management increases since renewal decisions have to be made.
If a more advanced renewal decision rule is used, communication between the lessor and lease component is required (see below).

Application of a renewal policy seems a reasonable choice for most application scenarios due to its additional flexibility. Note that, depending on the other components of a leasing variant, a renewal policy can also be the right choice for time-critical applications and in scenarios where a high degree of fairness is desired (see Section 4).

Contrary to a no renewal policy, a renewal policy seems a reasonable choice if

- resources are lightly utilized
- the majority of tasks are difficult to subdivide and/or cannot be performed in a single lease period

3.4. Renewal Decision Rules

The renewal decision rule determines under which circumstances a renewal request is granted. An example of a renewal decision rule is the following rule: A lease claimant is granted a lease renewal as long as the total lease time of this claimant does not exceed a certain time limit. If simple renewal decision rules, like e.g. the rule stated above, are used, the lease component is capable of making the renewal decision, i.e. no communication with the lessor is necessary. For all advanced renewal decision rules, e.g. if the renewal decision depends on properties of the applications in the waiting queue, the lease has to communicate the renewal request to the lessor, which decides whether the renewal is granted or not.

Since the applicability and the effects of most renewal decision rules depend on the choice of the other three components of a leasing variant, we will introduce and discuss them during the following analysis of leasing variants.

4. Analysis of Leasing Variants

In order to make the following discussion more concise, we introduce the following definition of a leasing variant.

A leasing variant is a quadruple \([r, s, t, u]\) consisting of:

1. a request processing policy \(r \in \{\text{no queue, queue}\}\)
2. a scheduling algorithm \(s \in \{\text{FCFS, priority}\}\)
3. a renewal policy \(t \in \{\text{renewal, no renewal}\}\)
4. a decision rule \(u \in R\), where \(R\) denotes the set of available decision rules.

For example, \([\text{queue, FCFS, no renewal, -}]\) represents a leasing variant with a waiting queue managed as FCFS and a no renewal policy. A - entry indicates that the entry is of no use in the considered variant, for example, the - entry for the decision rule in the above example indicates that no decision rule is needed with a no renewal policy. We use an asterix as tuple entry to indicate that all possible values for this entry are addressed in the discussion of the variant. The following analysis of leasing variants is organized according to the applied lease renewal policy.

4.1. Leasing Variants with a No Renewal Policy

We differentiate the following leasing variants with a no renewal policy:

- \([\text{no queue, - , no renewal, -}]\) Combining a no renewal policy with a no queue policy strongly emphasizes the fairness aspect and contributes to an effective resource management, since each application can occupy the resource for at most one leasing period before other applications get a chance of acquiring a lease. Moreover, the leasing process is kept as simple as possible, since no renewal decisions have to be made and no waiting queue has to be managed. The disadvantages are identical to the general disadvantages of a no renewal policy described above.

- \([\text{queue, *, no renewal, -}]\) In case a no renewal policy is combined with a queue policy, we get exactly the general advantages and disadvantages of a no renewal policy described above. Using FCFS scheduling additionally achieves a high degree of fairness. Usage of a priority-based scheduling algorithm, e.g. SRT, additionally yields the advantages of the scheduling algorithm, e.g. minimal average waiting times but faces the problem of starvation.

4.2. Leasing Variants with a Lease Renewal Policy

If a renewal policy is combined with a no queue policy, we differentiate two different renewal decision rules, total lease time and interim lease denials. We discuss them in the context of the resulting leasing variant:

- \([\text{no queue, -, renewal, total lease time}]\) The first possibility of a renewal decision rule is to set a limit for the total lease time. The total lease time is defined as the time a claimant uninterruptedly holds a lease including possible renewals. If an application has already occupied the resource for a time period greater or equal than this total lease time, a renewal request is denied, otherwise
the renewal request is granted. No communication between the lease and lessor component is necessary if this decision rule is applied. Therefore, the resulting leasing variant is straightforward to realize. The quality of this leasing variant depends on the tuning of the upper time limit.

Setting the upper time limit rather high has the advantage that the majority of applications are able to finish their tasks during the total lease time and therefore do not have to subdivide their tasks. The disadvantage of a high time limit is the fact that the resource management becomes ineffective and the leasing rather unfair since other claimants might have to wait a long time before getting the chance of acquiring a lease. For this reason, to allow for unrestricted prolongation of the usage time as proposed by some authors endangers the benefits of the leasing approach and is therefore discouraged by the authors of this paper. On the other hand, setting the upper time limit rather low yields a high degree of fairness and an effective resource management, while possibly forcing many an application to subdivide its task because the total lease time is too short.

**[no queue, -, renewal, interim lease denials]** A second possibility of a renewal decision rule is to base the renewal decision on the number of times other claimants requested a lease for the resource during its total lease time. For example, a claimant may be granted renewals until five other claimants have expressed interest in the resource. Then, the next renewal request is denied. To realize this approach, the lease component has to communicate with the lessor component in order to decide whether the renewal is to be granted. This is because information about the number of interim lease denials is only known to the lessor component. The quality of this variant depends on the tuning of the maximal value of interim lease denials. Setting the maximal value of interim lease denials must consider how strongly the resource is used, i.e. the quality of the leasing variant really depends on the ratio between the value of interim lease denials and resource usage. If the value of denials is rather high in comparison to the expected resource usage, we get the same pros and cons as when setting a high upper time limit in the above variant, i.e. most claimants do not have to subdivide their tasks but fairness and effectiveness of resource management are rather poor. Analogously, setting a rather small value of denials in comparison to the expected resource usage will yield the same results as setting a small upper time limit in the above variant, i.e. many claimants might have to subdivide their tasks while fairness and effectiveness of resource management are strongly emphasized.

If a renewal policy is combined with a queue policy, we examine two different renewal decision rules, **no waiting claimant** and **priority**. Again, we discuss them in the context of the resulting leasing variant:

**[queue, *, renewal, no waiting claimant]** The renewal decision rule **no waiting claimant** means that a renewal is only granted if the waiting queue for the affected resource is empty, i.e. no other claimant is interested in a lease for the resource. Obviously, communication between the lease and lessor component is required for this variant.

If FCFS scheduling is used to manage the waiting queue, a high degree of fairness and effective resource management are guaranteed by this variant since a renewal request is denied whenever another application is also interested in a lease (and therefore waiting in the waiting queue). However, facing a high resource usage this variant might force lots of applications to subdivide their tasks since they are only granted one lease period a time.

If a priority scheduling algorithm is used, applications that are denied a renewal request can pass other applications in the waiting queue after being again added to the waiting queue. Therefore, fairness is no longer guaranteed and typical problems like starvation can occur while the disadvantage of forcing applications to subdivide their tasks remains. Moreover, it is difficult to anticipate how strong higher-priority applications are favored over lower-priority ones. This is due to the fact that the degree of privilege of higher-priority applications depends on how long a lease claimant that has been denied a renewal request has to wait before again being added to the waiting queue. Therefore, we discourage combining a priority scheduling algorithm with the **no waiting claimant** decision rule.

**[queue, *, renewal, priority]** The renewal decision can be based on a comparison of the priorities of the application currently holding the lease and the next one to be scheduled, i.e. the one on the first position in the waiting queue. In this way it is possible to free applications from subdividing their tasks by assigning them a high priority that ensures that they are granted the required total lease time. Obviously, the fairness aspect suffers if the renewal decision is based on priorities.

If an FCFS scheduling algorithm is used, the renewal decision only depends on the priority of the application requesting the renewal and that of the first application in the waiting queue. The following example shows why this may pose problems. Imagine A to be the application currently holding the lease \(\text{priority}(A) = 7\), application B is the first application in the waiting queue \(\text{priority}(B) = 5\) and application C comes...
second \(\text{priority}(C) = 9\). Application \(A\)'s renewal request is granted, although a higher-priority application, namely \(C\), is waiting for the resource. This contradicts the intention behind using a priority-based decision rule.

Using a priority scheduling algorithm based on the same priorities as the renewal decision seems a more reasonable choice. In this case, the renewal decision is only granted if the requesting application has a higher priority than all applications in wait. For the above example, this means that \(A\) is not granted the renewal since \(C\) is first in line and \(C\) has a higher priority than \(A\). Obviously, this variant has the disadvantage that it is possible that a waiting application might be starved. However, the aging concept can remedy this situation.

One example of a priority-based renewal decision rule is to base it on the time that the resource is required by the application, i.e. analogous to SRT scheduling. If the requested renewal time is shorter than the time that the first application in the waiting queue requires the lease, the renewal is granted.

5. Conclusion

In this paper, we present a systematic classification of leasing variants by clearly defining the components of a leasing variant, namely the request processing policy, the applied scheduling algorithm, the renewal policy and the renewal decision rule. Contrary to the existing literature, we consider the possibility that claimants are able to wait for resources and thoroughly examine possible renewal decision rules and the effects of varying their parameters. We outline the advantages and disadvantages of the leasing variants resulting from our classification and point out possible fields of application.

As future work, we plan to enhance the leasing service introduced by [1, 2] to cover the above described relaxations, i.e. it has to be possible for a claimant to wait for a resource and the communication between lease and lessor necessary to implement more sophisticated renewal decision rules has to be realized.

References