

# Self-Adaptive Leasing for Jini

Kevin Bowers  
Rensselaer Polytechnic Institute  
bowerk@rpi.edu

Kevin Mills and Scott Rose  
National Institute of Standards and Technology  
{kmills, srose}@nist.gov

## Abstract

*Distributed systems require strategies to detect and recover from failures. Many protocols for distributed systems employ a strategy based on leases, which grant a leaseholder access to data or services for a limited time (the lease period). Choosing an appropriate lease period involves tradeoffs among resource utilization, responsiveness, and system size. We investigate these issues for Jini Network Technology. First, we establish quantitative tradeoffs among lease period, bandwidth utilization, responsiveness, and system size. Then, we consider two self-adaptive algorithms that enable a Jini system, given a fixed allocation of resources, to vary lease periods with system size to achieve the best responsiveness. We compare performance of these self-adaptive algorithms against each other, and against fixed lease periods. We find that one of the self-adaptive algorithms proves easy to implement and performs reasonably well. We anticipate that similar procedures could add self-adaptive capability to other distributed systems that rely on leases.*

## 1. Introduction

Distributed systems require strategies to detect and recover from failures. One commonly used strategy employs a leasing mechanism, where a node grants a leaseholder access to a resource for a limited time (the lease period). If the resource is needed beyond the original lease period, then the leaseholder can renew the lease by requesting additional lease periods. Once the resource is no longer needed, the leaseholder may relinquish its lease. If the leaseholder does not renew a lease before expiration of the lease period, the lease grantor assumes leaseholder failure and terminates the lease to prevent resource leaks. Since originally proposed by Gray and Cheriton for consistency maintenance in a distributed file cache [1], leases have become widely used in a range of applications [2-6].

In any leasing system, questions arise regarding how to select the lease period. Choosing an appropriate lease period requires consideration of tradeoffs among resource utilization, responsiveness, and number of leaseholders. We investigate these issues in the context of service-discovery protocols, which allow distributed software components to

discover each other and compose themselves into assemblies that cooperate to meet application needs. Though several service-discovery protocols currently exist [e.g., 5-8], we selected Jini Network Technology [5] for our study because leasing plays a central role in registering Jini services. We base our modeling and analysis on the Jini specification [7].

We investigate self-regulating algorithms for achieving the best available responsiveness from a leasing system as system size varies, while respecting a constraint on resources devoted to leasing. We begin by establishing quantitative tradeoffs among responsiveness, resource consumption, and system size. Then, we propose two different self-regulating algorithms for varying lease periods in response to changing system size. We use simulation to compare the effectiveness of the algorithms against each other and against fixed lease periods. We consider whether one of the algorithms might be used to improve performance of Jini leasing and discuss using the algorithm in other service-discovery protocols, such as Universal Plug-and-Play (UPnP) [6].

## 2. Jini Leasing

Jini defines an architecture that enables clients and services to rendezvous through a third party, known as a lookup service. A Jini service registers a description of itself with each discovered lookup service. A Jini client may register a request to be notified by a lookup service of arriving or departing services of interest, or of changes in the attributes describing services of interest.

Figure 1 illustrates message exchanges for some typical Jini leasing scenarios. A registering component requests registration for a duration ( $L_R$ ), which may be accepted at time  $T_G$  for a granted lease period  $L_G \leq L_R$ .  $L_R$  may be any, which allows any value for  $L_G$ . To extend registration beyond  $L_G$ , registering components must renew the lease prior to an expiration time  $T_E = T_G + L_G$ ; otherwise, registration is revoked. This cycle continues until a Jini component cancels or fails to renew a lease. Lookup services assign  $L_G$  within a configured range,  $L_{MIN} \leq L_G \leq L_{MAX}$ . While a granted lease may not be revoked prior to  $T_E$ , lookup services may deny any lease request. Jini components must adopt strategies for selecting values for  $L_R$ . Similarly, lookup services must determine algorithms for assigning values for  $L_G$ ,  $L_{MIN}$ , and

$L_{MAX}$ ; and for deciding when to deny leases. We identify some relevant relationships.

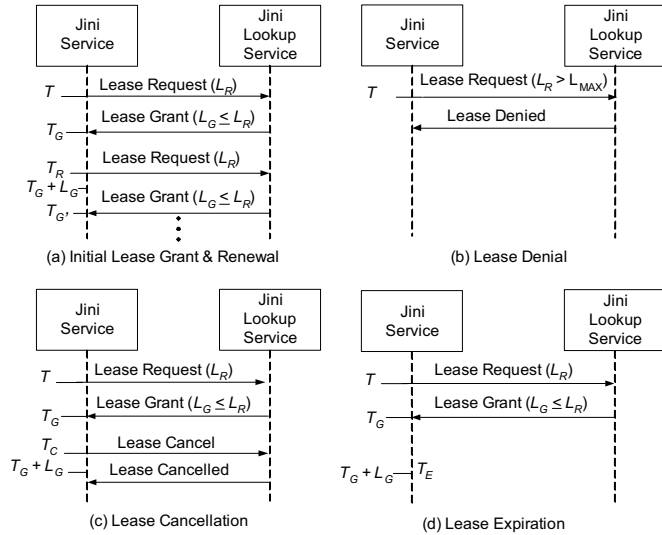


Fig. 1. Message exchanges for four Jini leasing scenarios.

Let  $S_R$  be lease-request size,  $S_G$  be lease-grant size, and  $N$  be the number of leaseholders. Typically, a leaseholder and lookup service exchange one request-grant pair per renewal cycle, with rate  $1/L_G$  Hz. Assuming identical  $L_G$  assigned for each lease, bandwidth use ( $B$ ) can be estimated as:  $B=(N/L_G) \cdot (S_R+S_G)$ . Assuming constant  $S_R$  and  $S_G$ ,  $B$  increases linearly with  $N$  and decreases exponentially with  $L_G$ . Another metric, responsiveness,  $R$ , measures the latency with which lookup services can detect leaseholder failure. Assuming uniformly distributed failure times, then expected responsiveness is  $R = L_G / 2$ ; thus,  $R$  is independent of  $N$ , but  $B$  and  $R$  are related through  $L_G$ .

These relationships can be used to constrain and predict behavior of a leasing system. For example, assume known requirements for  $R$  and  $B$ . The responsiveness equation can be rewritten to determine  $L_G$  [i.e.,  $L_G=2R$ ]. Then, using  $L_G$ , the bandwidth equation can be transformed to find maximum system size [i.e.,  $N_{MAX}=(B \cdot L_G)/(S_R+S_G)$ ]. With this information, lookup services could grant lease periods  $\leq L_G$  to ensure required responsiveness, deny requested leases that would consume an excess share of bandwidth, and deny requests for leases once  $N$  reaches  $N_{MAX}$ .

### 3. Two Self-adaptive Leasing Schemes

We consider two techniques to vary  $L_G$  with  $N$ ; thus, using available bandwidth ( $B$ ) to achieve the best possible responsiveness ( $R$ ) for a given value of  $N$ . One technique restricts lease requests to  $L_R = any$ . The second technique inverts the leasing process, permitting lookup services to poll leaseholders at a variable interval.

*Restricting  $L_R$ .* Assuming a leasing system must consume at most bandwidth  $B$  and guarantee minimum average responsiveness  $R_{MIN}$ , a lookup service can grant a maximum lease period  $L_{MAX} = 2R_{MIN}$ . Given  $B$ ,  $S_R$ , and  $S_G$ , we can determine a maximum lease-renewal rate  $G = B / (S_R + S_G)$ . For minimum system size,  $N_{MIN} = 1$ , the lookup service can grant a minimum lease period  $L_{MIN} = 1/G$ . While this value for  $L_{MIN}$  respects the bandwidth constraint, other factors should be considered. For example, at  $L_{MIN} = 1/G$  leaseholder processing burden might prove unacceptable. Instead, a leasing system might constrain maximum responsiveness ( $R_{MAX}$ ), giving a minimum lease period  $L_{MIN} = 2R_{MAX}$ . Knowing  $N$ , a lookup service may select a suitable granted lease period from a range ( $L_{MIN} \leq L_G \leq L_{MAX}$ ) using a simple algorithm. First, compute  $L_G = N/G$ . If  $L_G > L_{MAX}$ , then deny the lease; otherwise, if  $L_G < L_{MIN}$ , then set  $L_G = L_{MIN}$ . Assigning  $L_G$  with this algorithm permits a leasing system to constrain  $B$  and guarantee minimum average responsiveness ( $R_{MIN}$ ), while providing the best responsiveness achievable (up to  $R_{MAX}$ ) as  $N$  varies over  $1..N_{MAX}$ .

*Inverted Leasing.* As an alternative, we could invert the leasing process so that a lookup service polls periodically on a multicast channel, where all leaseholders listen. Figure 2 illustrates some associated message exchanges. To obtain a lease, a leaseholder sends (via reliable unicast) a lease request to the lookup service, which returns a time ( $T_p$ ) when the leaseholder should expect to hear a multicast poll.

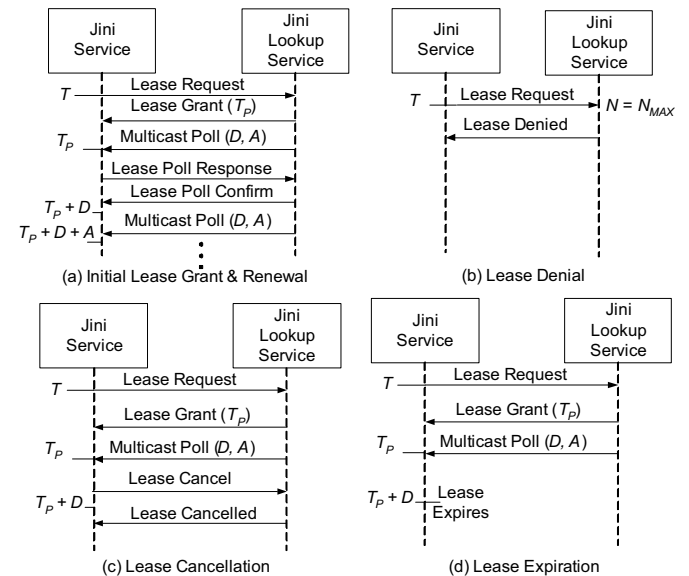


Fig. 2. Message exchanges for inverted leasing mechanism.

Each poll includes two values: the duration ( $D$ ) over which the lookup service will listen for leaseholders to respond and the additional time ( $A \geq 0$ ) beyond  $D$  within which leaseholders can expect the next poll. Each leaseholder chooses a random time (distributed uniformly over  $0..D$ ) to respond to the lookup service, which confirms each response.

The lookup service cancels a lease if the leaseholder does not respond within  $D$ . Similarly, failing to receive a poll within  $D + A$  after the previous poll, causes a leaseholder to request a new lease. The main issue is selecting values for  $D$  and  $A$  in each poll.

Assuming the polling interval is bounded by  $L_{MIN} \leq D + A \leq L_{MAX}$ , the lookup service computes  $D = \max(N/G, L_{MIN})$ . A rapidly expanding system might benefit from deferring the next poll until  $D + A$  to accommodate increases in  $N$  during  $D$ . Choosing an appropriate value for  $A$  depends on system growth expected during  $D$ . In our experiments, we set  $A$  as a percentage of  $D$ . Recall, though, that  $D + A \leq L_{MAX}$ , so  $A$  may be reduced below its computed value. When  $A = 0$ , the leasing system has reached maximum capacity. To ensure this, the lookup service must deny lease requests that will cause  $N$  to exceed  $N_{MAX}$ , where  $N_{MAX} = L_{MAX} \cdot G$ .

When using inverted leasing, a lookup service limits bandwidth usage according to  $B = S_P + ((N/P) \cdot (S_{PR} + S_{RC}))$ , where  $P$  is the polling interval ( $D < P \leq D + A \leq L_{MAX}$ ) and  $S_P$ ,  $S_{PR}$ , and  $S_{RC}$  represent respectively the size of poll, poll-response, and response-confirm messages. Inverted leasing achieves system responsiveness of  $R = D$ , which is only  $1/2$  as responsive as simple adaptive leasing. To understand this difference, consider the following analysis.

Assume failure times are distributed uniformly on  $D$ . Failures may occur either before or after a leaseholder responds to a poll. For leaseholders that fail before a poll, expected failure-detection latency is  $D/2$ . For leaseholders that fail after a poll, expected failure-detection latency increases to  $(D/2) + D$ . Assuming that failures are equally likely before or after a poll, then  $R = 1/2 \cdot (D/2) + 1/2 \cdot (3D/2)$ , which reduces to  $R = D$ .

#### 4. Simulation Results and Discussion

We used simulation to investigate dynamic behavior of our self-adaptive algorithms. We coded an SLX discrete-event simulation [9] model of Jini. To confirm our analysis and to verify our simulation, we conducted simulation experiments, varying  $N$  from 10..200 and  $L_G$  from 15..300 s in 15-s increments. We used  $S_R = 128$  bytes and  $S_G = 32$  bytes. Figure 3 shows simulated results for average  $B$  and  $R$  when  $L_G = 15$  s, 60 s, and 120 s. Our simulation confirms our analyses: (1)  $B$  increases linearly with  $N$  for a given  $L_G$  and decreases exponentially with  $L_G$  for a given  $N$  and (2)  $R = L_G/2$ , independent of  $N$ .

Next, we created model variants to implement the self-adaptive leasing algorithms described in Section 3. One variant (Adaptive) replaces fixed  $L_G$  with our simple adaptive algorithm; the other variant (Inverted) substitutes our inverted procedures for Jini leasing. We measured  $B$  under increasing and decreasing  $N$ . We measured the control variable ( $L_G$  for Adaptive and  $D$  for Inverted) under

increasing  $N$ , and we measured  $R$  under decreasing  $N$ . We set  $L_{MIN} = 15$  s,  $L_{MAX} = \infty$ , and  $G = 3$ . For experiments involving Inverted, we set  $A = 0.2 \cdot D$ .

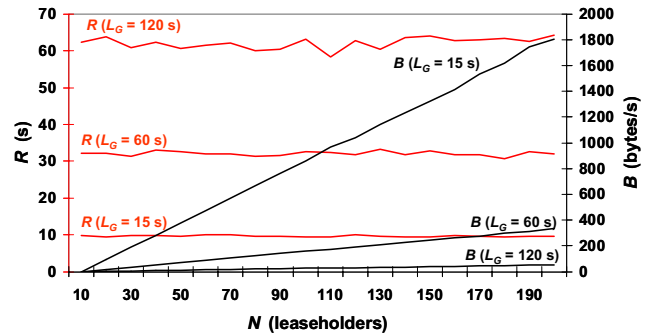


Fig. 3. System responsiveness ( $R$ ) – left-hand y-axis – and bandwidth usage ( $B$ ) – right-hand y-axis – for three granted lease periods ( $L_G = 15$  s, 60 s, and 120 s) as system size increases ( $N = 10$  to 200 leaseholders).

Figure 4 depicts both Adaptive and Inverted under increasing  $N$ . While the control variables change in a similar fashion, change in  $B$  exhibits two obvious differences. First,  $B$  increases more steeply under Adaptive than under Inverted. Second, Inverted begins to constrain  $B$  earlier than Adaptive, which leads to a higher peak bandwidth usage. Inverted affects all leaseholders with each adjustment in the control variable, while Adaptive affects leaseholders one-by-one, and only as each lease is renewed.

Figure 5 plots average  $R$  achieved by each self-regulating scheme as  $N$  decreases. Inverted begins to reduce  $B$  sooner than Adaptive. For  $R$ , the results tell two stories. First, as indicated by a steeper negative slope, Inverted adapts  $R$  more quickly than Adaptive. Unfortunately, Inverted achieves only  $1/2$  the responsiveness of Adaptive. Implementing Inverted would require profound changes in Jini. Adaptive can be implemented easily within Jini lookup services, and might apply to domain-wide leasing.

Each Jini service is required to register its service description with each appropriate lookup service that it discovers; thus, a service may be maintaining leases on  $N_D$  different lookup services. System-wide leasing demands will vary with  $N_D$ . Assuming a known network-wide resource budget for leasing, e.g., either aggregate bandwidth ( $B_D$ ) or renewal rate ( $G_D$ ), then each lookup service can compute its share (either  $B_D/N_D$  or  $G_D/N_D$ ). Jini facilitates monitoring  $N_D$  by requiring each lookup service to announce itself periodically. By monitoring announcements, each lookup service can increment and decrement  $N_D$  as lookup services come and go, and continuously adjust its share of resources.

Our results might also apply to a number of leasing schemes outside of Jini. For example, UPnP devices manage variables for which they may offer subscriptions to control points. UPnP subscription procedures, and associated

parameters, appear quite similar to those defined in Jini. We are confident our adaptive leasing algorithm could be applied to UPnP, yielding performance properties similar to those we report for Jini.

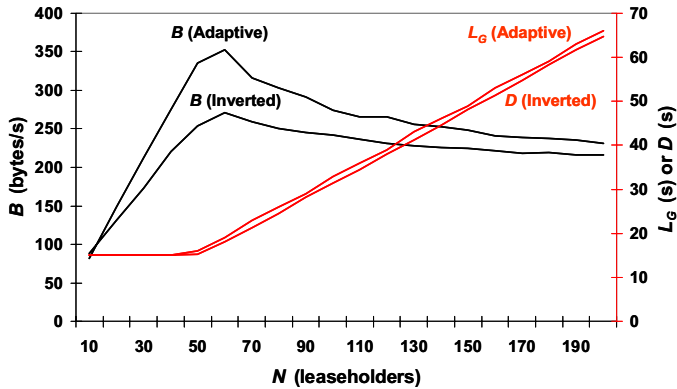


Fig. 4. Bandwidth usage ( $B$ ) – left-hand y-axis – and control variable ( $L_G$  for Adaptive and  $D$  for Inverted) setting – right-hand y-axis – as system size increases ( $N = 10$  to  $200$  leaseholders).  $L_{MIN} = 15$  s,  $G = 3$  renewals per second,  $L_{MAX} = \infty$ , and (for Inverted)  $A = 0.2D$ .

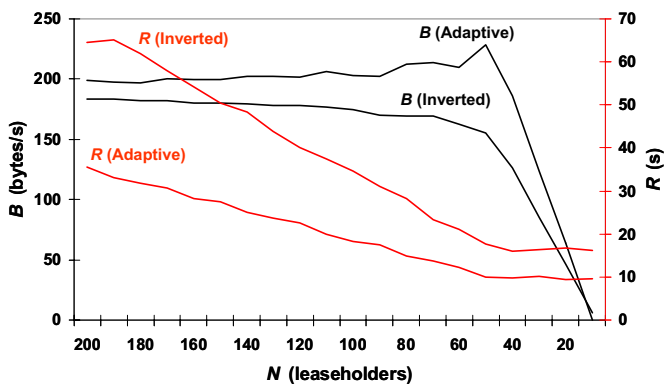


Fig. 5. Bandwidth usage ( $B$ ) – left-hand y-axis – and system responsiveness ( $R$ ) – right-hand y-axis – as system size decreases ( $N = 200$  to  $0$  leaseholders).  $L_{MIN} = 15$  s,  $G = 3$  renewals per second,  $L_{MAX} = \infty$ , and (for Inverted)  $A = 0.2D$ .

## 5. Conclusions

We investigated Jini leasing procedures, establishing quantitative tradeoffs among responsiveness, resource consumption, system size, and granted lease period. We suggested an approach to bound bandwidth use, while guaranteeing a minimum level of responsiveness in detecting leaseholder failures. We also showed a simple adaptive leasing algorithm that bounds bandwidth consumption, while achieving the best available responsiveness as system size varies. We described an alternate algorithm that inverts the leasing process, and we showed that inverted leasing

achieves only half the responsiveness guaranteed by the simple adaptive algorithm. We used simulation to show that inverted leasing adapts responsiveness more quickly and constrains bandwidth consumption better than our simple adaptive algorithm. Given the performance tradeoffs and implementation costs, we conclude that our simple adaptive leasing algorithm can yield useful performance properties with little cost. We outlined a simple technique for allocating a domain-wide resource budget among multiple lease grantors. We expect our analyses can be used to deploy Jini systems with understood leasing behavior, and we hope our ideas for adaptive leasing can provide improvements over static strategies. We argued that our adaptive leasing algorithm and related analyses should also apply in similar leasing systems, such as event subscriptions offered by UPnP.

## 6. References

- [1] C. Gray and D. Cheriton. “Leases: an efficient fault-tolerant mechanism for distributed file cache consistency”, *ACM SIGOPS Operating Systems Review, Proceedings of the Twelfth ACM symposium on Operating systems principles*, November 1989, Volume 23 Issue 5.
- [2] Charles E. Perkins and Kevin Luo. “Using DHCP with computers that move”, *Wireless Networks*, March 1995, Volume 1 Issue 3.
- [3] Anoop Ninan, Purushottam Kulkarni, Prashant Shenoy, Krithi Ramamritham, and Renu Tewari. “Performance: Cooperative leases: scalable consistency maintenance in content distribution networks”, *Proceedings of the eleventh international conference on World Wide Web*, May 2002.
- [4] Jacob Harris and Vivek Sarkar. “Lightweight object-oriented shared variables for distributed applications on the Internet”, *ACM SIGPLAN Notices, Proceedings of the conference on Object-oriented programming, systems, languages, and applications*, October 1998, Volume 33 Issue 10.
- [5] Jim Waldo. “The Jini™ architecture for network-centric computing”, *Communications of the ACM*, July 1999.
- [6] [Universal Plug and Play Device Architecture](#), Version 1.0, 08 Jun 2000 10:41 AM. © 1999-2000 Microsoft Corporation. All rights reserved.
- [7] Ken Arnold et al, [The Jini Specification](#), V1.0 Addison-Wesley, 1999. The latest version is available on the web from Sun.
- [8] Service Location Protocol Version 2, Internet Engineering Task Force (IETF), RFC 2608, June 1999.
- [9] James O. Henriksen, “An Introduction to SLX™”, *Proceedings of the 1997 Winter Simulation Conference*, ACM, Atlanta, Georgia, December 7-10, 1997, pp. 559-566.