Improving Failure Responsiveness in Jini Leasing

Scott Rose, Kevin Bowers, Steve Quirolgico, and Kevin Mills
National Institute of Standards and Technology
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 explain these tradeoffs for Jini Network Technology. Then, we describe an adaptive algorithm that enables a Jini system, given a fixed allocation of resources, to vary lease periods with system size to achieve the best responsiveness. We anticipate that similar procedures could improve failure responsiveness in other distributed systems that rely on leases. We describe how we implemented our adaptive algorithm in "reggie", a publicly available implementation of the Jini lookup service. We can use our implementation to demonstrate how adaptive leasing provides the best available responsiveness as network size varies.

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). Once the resource is no longer needed, the leaseholder may relinquish its lease. If the resource is needed beyond the original lease period, then the leaseholder can renew the lease by requesting additional lease periods. If the leaseholder does not renew before expiration of the lease period, the lease grantor assumes leaseholder failure and terminates the lease.

Choosing an appropriate lease period entails tradeoffs among resource utilization, responsiveness, and number of leaseholders. We explore these issues in the context of service-discovery protocols, which allow distributed software components to discover each other and compose themselves into assemblies. Though several service-discovery protocols currently exist [e.g., 1-4], we selected Jini Network Technology [1] to demonstrate our ideas, because leasing plays a central role in registering Jini services. We base our analysis on the Jini specification [2].

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.

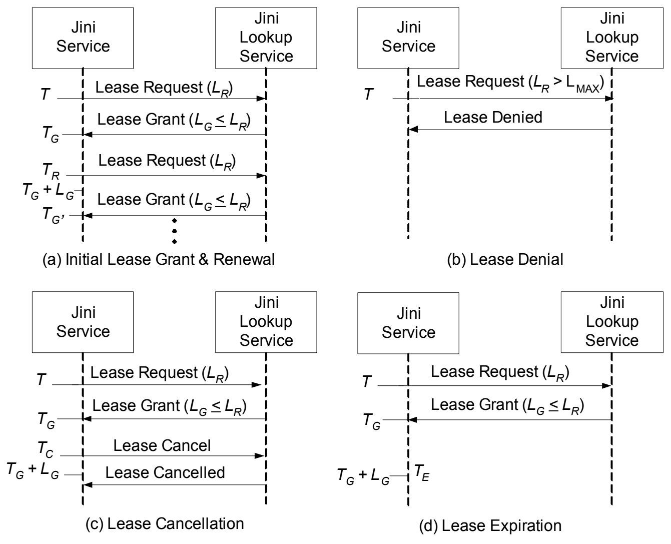


Fig. 1. Message exchanges for four Jini leasing scenarios.

Figure 1 illustrates message exchanges for some typical Jini leasing scenarios. A registering component requests registration for 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.

We can analyze performance of a Jini leasing system. Let S_R be lease-request size, S_G be lease-grant size, and N be 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}.

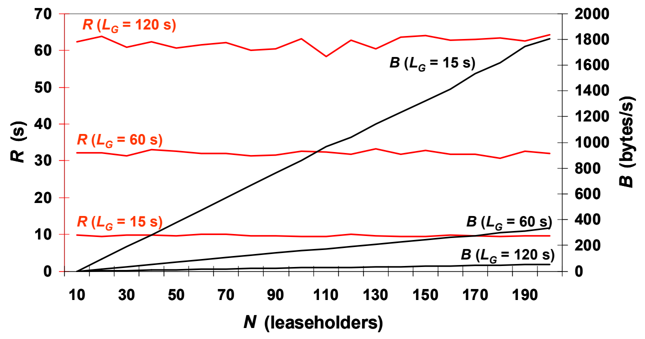


Fig. 2. System responsiveness (*R*) – left-hand y-axis – and bandwidth usage (*B*) – right-hand y-axis – for three granted lease periods (*LG* = 15 s, 60 s, and 120 s) as system size increases (*N* = 10 to 200 leaseholders).

3. A Self-adaptive Algorithm for Jini Leasing

We propose an algorithm that restricts lease requests to L_R = *any*. 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 $1..N_{MAX}$.

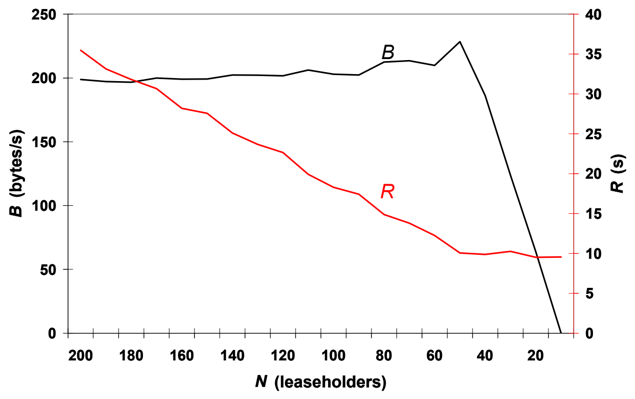


Fig. 3. Responsiveness (*R*) – left-hand y-axis – and bandwidth usage (*B*) – right-hand y-axis –as system size decreases (*N* = 200 to 0 leaseholders).

4. Simulation Results

We coded an SLX discrete-event simulation [5] model of Jini to confirm our analysis and to investigate dynamic behavior of our self-adaptive algorithm. 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 and S_G = 32 bytes. Figure 2 shows simulated results for average B and R when L_G = 15 s, 60 s, and 120 s. The 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 simulated our adaptive leasing algorithm. Figure 3 illustrates how the algorithm constrains B while improving R as system size decreases. These promising results led us to implement our adaptive algorithm in "reggie", a publicly available implementation of a Jini lookup service.

5. Implementation in "reggie"

We base our adaptive-leasing implementation on the "reggie" lookup server provided with the Sun Microsystems Jini release. The "reggie" server implements the Jini specification for a lookup service, and includes additional extensions to allow remote administration of the lookup server through a service proxy. Administrative actions occur through the *RegistrarAdmin* interface, which is not part of the

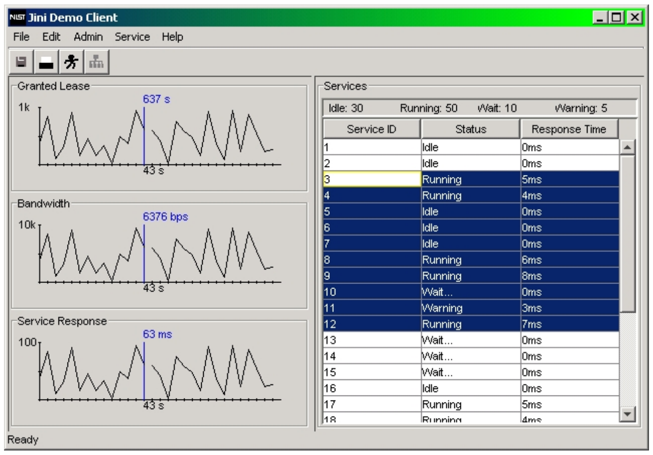
Jini core specification, but a Sun extension to Jini (**com.sun.jini.reggie** package). The *RegistrarAdmin* interface allows basic monitoring, configuration, and control of an operational lookup server just as if it were any other type of Jini-enabled service. Using the *RegistrarAdmin* interface, an administrator can also perform some basic manipulation of minimum and maximum granted lease periods for services and events maintained on the lookup server. However, this method requires constant human supervision to optimize leasing performance in a Jini network. Such human supervision would prove impractical in a large network where numerous services may join and leave.

To implement adaptive leasing, we modified the "reggie" server implementation to assign lease grant times (L_G) based on the required failure responsiveness (R) of the system and the bandwidth (B) allocated to lease renewal transactions. We added a collection of access methods to the *RegistrarAdmin* interface, callable via remote-method invocation (RMI), allowing a Jini client to view: the current L_{MIN}, L_{MAX}, and L_G, the number of leaseholders (N) on the server, the instantaneous average bandwidth (B_{AVG}) consumed by lease renewals, and the instantaneous average failure responsiveness (R_{AVG}). Since values for granted lease periods can be adjusted from changes to the allocated bandwidth and target responsiveness, we added methods to set B and R in the *RegistrarAdmin* interface.

The lookup server (**com.sun.jini.reggie.RegistrarImpl**) starts with default values for L_{MIN} and L_{MAX}. A Jini client can use the *RegistrarAdmin* interface to adjust target responsiveness and allocated bandwidth. Based on these adjustments, the lookup server computes new values for L_{MIN} and L_{MAX}. At regular intervals, the lookup service samples average bandwidth use (B_{AVG}) and the number of leaseholders, adjusting granted lease periods (L_G) accordingly. Current bandwidth usage is calculated by multiplying the number of lease transactions (RMI calls) by the size of messages involved in the transaction. Currently, the lookup service records these values once every sixty seconds of operation, or when an administrator changes R or B.

When a Jini service registers with the lookup service, it may either request a specific lease interval or use Jini's *LEASE.ANY* constant to allow the lookup server to select an appropriate lease period for the service. In our implementation, if the service requests the *LEASE.ANY* constant, the lookup service uses the current value for L_G as the granted lease period. Otherwise, if the service requests a lease period in the range of L_{MIN} and L_{MAX}, it is granted. The lookup service rejects requests for leases outside this range.

Figure 4 shows a snapshot of a Jini client graphical user interface (GUI) that uses the modified "reggie" *RegistrarAdmin* interface. The left-hand column plots values for L_G, for bandwidth used (B_{AVG}), and for average responsiveness (R_{AVG}) over time. These graphs display values returned to the client from regular polling of access methods *RegistrarAdmin* interface, which uses RMI to call the corresponding method in the lookup server. The right-hand column of Figure 4 lists leaseholders using the lookup service, and displays their current status in the Jini network. Note that the GUI displays only the current *known* status, as a service may have left the network, but a proxy could still be registered with the server. The proxy would be purged when its lease expires. The GUI in Figure 4 does not include the *RegistrarAdmin* GUI used to adjust allocated bandwidth or target responsiveness in the lookup service.



Idle: 30 Running: 50 Wait: 10 Warning: 5

Service ID	Status	Response Time
1	Idle	0ms
2	Idle	0ms
3	Running	5ms
4	Running	4ms
5	Idle	0ms
6	Idle	0ms
7	Idle	0ms
8	Running	6ms
9	Running	8ms
10	Wait...	0ms
11	Warning	3ms
12	Running	7ms
13	Wait...	0ms
14	Wait...	0ms
15	Wait...	0ms
16	Idle	0ms
17	Running	5ms
18	Running	4ms

Fig. 4. Sample Graphical User Interface for a Jini client monitoring a lookup server that implements adaptive leasing

6. Acknowledgments

The work described in this paper is funded in part by the DARPA Fault-Tolerant Networks program and the NIST Pervasive Computing program. We gratefully acknowledge support from Dr. Doug Maughan of DARPA and Dr. Susan Zevin of NIST.

7. References

[1] Jim Waldo. "The Jini^{TM} architecture for network-centric computing", *Communications of the ACM*, July 1999.

[2] Universal Plug and Play Device Architecture, Version 1.0, 08 Jun 2000 10:41 AM. © 1999-2000 Microsoft Corporation. All rights reserved.

[3] Ken Arnold et al, The Jini Specification, V1.0 Addison-Wesley, 1999. The latest version is available on the web from Sun.

[4] Service Location Protocol Version 2, Internet Engineering Task Force (IETF), RFC 2608, June 1999.

[5] James O. Henriksen, "An Introduction to SLX^{TM}" *Proceedings of the 1997 Winter Simulation Conference*, ACM, Atlanta, Georgia, December 7-10, 1997, pp. 559-566.