Assignment schemes for replicated services in Jini.

Vasil Georgiev
Vladimir Getov
Harrow School of Computer Science

Assignment Schemes for Replicated Services in Jini*

Vasil Georgiev and Vladimir Getov
School of Computer Science, University of Westminster
Northwick Park, Harrow HA1 3TP, U.K.
E-mail: {V.Georgiev,V.S.Getov}@wmin.ac.uk

Abstract

This paper introduces and compares different schemes for assignment of replicated services in Jini – an object-oriented middleware architecture for network-centric computing. Each client in Jini has to be assigned a service selected from the pool of available services, which have joined the Jini federation and registered with the lookup service. Both early and delayed assignments are considered as basic options in our evaluation. The information for the system load can be collected at four different levels of detail in order to be used in the process of assignment decisions. In our analysis, we concentrate on the scenario where the requests for service generated by the clients follow independent user-initiated or machine-initiated transactions. The performance evaluation of the assignment schemes follows the queuing systems methodology. The comparisons are done with regard to the mean residence time of the clients in the system as well as the control overhead imposed by the assignment schemes. A case study of the scheme using the lowest information level proves the effectiveness, applicability and limitations of the delayed assignment in comparison to the early one. These results are a first step towards developing a methodology for building large-scale applications for Jini-based distributed systems.

1. Introduction

1.1 Jini as middleware in Java-centric distributed processing

Jini is an object-oriented middleware architecture for network-centric computing. It provides a portable Java-centric mechanism that enables plugging together the distributed system components into a federation. The members of a Jini federation share services during the process of completing their tasks. The communication is based on a set of Java interfaces, which act as service protocols. The services themselves are located and assigned to the clients by a special lookup service. Thus, the federation components can be identified as services including one or more lookup service, and clients. The service protocols provide the necessary functionality for:

- registering the available services with the lookup service (the discovery and join protocols);
- service assignment on clients' demand (the lookup protocol);
- service access (via a specific service invocation protocol).

The most frequent example of a Jini system is probably the scenario in which a pool of printers is used by a set of clients over the network. However, presently the Jini technology has applications far broader than the device-connection technology toward which it was targeted originally [15]. In principle, any electronic device running a Java virtual machine (JVM) – so called intelligent device – could indeed replace the printers in the above scenario. Clusters of workstations and high-performance platforms do not commit any exception in this sense. Moreover, the Jini technology possesses features that substantially facilitate high-performance distributed processing:

- portability/mobility/native legacy binding [4],
- leasing (time-driven control for reliability and garbage collection [6]),
- distributed event-driven control [11], which somewhat resurrects the classic concept of communicating sequential processes.

These advantages and new features of Jini-based distributed platforms have attracted active interest by the developers of distributed applications. In Jini the transition between coding the drivers for remote (i.e. network-wide/federation-wide) intelligent devices and coding for remote task evaluation virtually dissolves or at least appears

*This work has been supported in part by HEFCE (UK) under the NFF initiative.

Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (EUROMICRO-PDP'02)
1066-6192/02 $17.00 © 2002 IEEE

to be rather spontaneous. However, from a performance point of view distributed data processing in a Jini federation raises the traditional questions about granularity, scalability and load balancing. In the case of printers and various other intelligent gadgets one can hardly foresee that a federation would comprise more than a few similar devices. At the same time, a distributed Jini-based platform can offer much greater number of similar or replicated services [1] - potentially as big as (if not greater than) the number of client processes. The availability of replicated services means that there are different options for assignment of services to clients. Furthermore, the lookup services in a federation (when more than one) may require hierarchical assignment schemes.

In this paper we focus on the scenario in which the Jini federation is a set of resources offering one or more types of replicated services. The resources are connected in a network thus forming a distributed system. The latter is loaded (and/or has been entered permanently) by a dynamic population of clients that seek the assignment of any of the available replicated services in the Jini federation. We consider here the service assignment task as a case of the general problem of load balancing in distributed systems. Normally, the objective of such a task is to minimise the residence time of the clients in a system via minimisation or elimination of the idle periods of the service resources. We show how and which of the known general schemes for load balancing are applicable and productive in the specific case of Jini.

1.2 General load balancing schemes

Here we present a compact taxonomy of general load-balancing approaches, which serve as a starting point for the formulation of Jini-specific assignment schemes. Following previous related work [5, 12, 14], one can identify two main branches of load balancing. The static load balancing (thoroughly covered in [5]) is suitable for clients arriving by group i.e. a distributed execution of a suitably decomposed application. The dynamic load balancing is designed to handle clients' jobs for which not only the amount of work but also the moments of arrival are subject of approximation. That is why dynamic load balancing requires a somewhat more elaborate operating structure. Admittedly, it consists of three recognisable and, in a sense, independent procedures often called polices [12]:

- information policy - this procedure gathers dynamic system information which is necessary when future load balancing decisions are to be taken;
- location policy - which consists of decision taking about the dynamic assignment of jobs to the available resources;
- transfer policy - this procedure comprises of the job transfer protocols.

Another taxonomy axis is the distribution of these three types of policies amongst the system entities (nodes). In this case, one can recognise three basic schemes - centralised, distributed and hybrid. In a Jini federation the assignment process is focussed on the lookup service. Actually, the standard Jini lookup service performs a centralised information-location policy. More specifically, the Jini lookup information policy consists of the information about the availability and the leasing conditions of the services. The job transfer is virtually not recognised within Jini architecture as it is left to the services themselves to deal (or rather not) with further dynamic redistribution of clients.

From a load balancing point of view, a distributed system is thought as a collection of sources (or senders) of tasks (e.g. the overloaded nodes) and receivers of tasks (e.g. underloaded and particularly idle nodes). Thus, load balancing recognises three types of balancing initiatives [14]:

- receiver-initiated balancing - the balancing decisions in regard of location policy are taken in the instants of change in the status of receivers e.g. in the moments of job's departure after being serviced when the node may become idle or at least underloaded to some criteria;
- sender-initiated balancing - symmetrically to the former - decisions are taken in the moments of job's arrival when the node become overloaded, and
- hybrid-initiated balancing - with obvious semantics.

The last taxonomy axis of consideration is the system information level, which serves as a decision basis for the resource (i.e. service) allocation. This axis may start with zero information schemes and "blind" decision-making - e.g. random allocation of the clients to the available services - and may extend to schemes based on full actual information about the system load status - e.g. monitoring of the local queue lengths of the services.

The above already available and well known load balancing concepts can be used as a basis for the development of performance-effective methods for distributed processing of asynchronous clients in Jini. The most direct approach is to extend the functionality of the lookup service to something like a component configurator [9]. For example, one can modify the process of client-service assignment by rescheduling the decision instants and collecting dynamically information about the system load in some more deatil.

The rest of this paper is organised as follows. Section 2 presents the system architecture, which is subject of modelling and analysis. Section 3. introduces the assignment

schemes, which can be employed by a distributed Jini application. The elements of these schemes are then presented in terms of queuing system modelling in section 4. Section 5. comprises a formal case study which serves both as a proof-of-concept for some of our performance claims and as a first step towards developing a methodology for analysis and design of Jini assignment schemes. The conclusion in section 6. summarises possible future work on the subject.

2. System architecture

The Jini architecture federates a network of autonomous devices (called in general nodes) running the JVM. The network forms a distributed Java-platform, which is homogenous in the sense that it provides a single execution environment and a downloaded code behaving in the same way in any of the nodes [2]. However the system architecture does not require homogeneity from a performance point of view: a method can be completed within substantially different time periods on different nodes thus making the system heterogeneous in the terms of our study. By convention Jini community consists of services, which are Java programming objects residing in and performed by the network nodes. Here we mean a set of replicated services, which do not differ in anything else beside their node of residence.

The nodes are permanent and persistent components of the Jini architecture. Their numbers do not vary. We accept that the system comprises N nodes (noted $n_1, ..., n_N$) running $S(t)$ replicated services $(s_1, s_2, ..., s_S)$. The number S may vary with time but normally $N = S$ though our considerations are not limited to this assumption. The case $N > S$ means that some of the nodes $(N - S)$ are performing transparent non-multiple unique services so they fall out of scope. $N < S$ would mean that a node is offering more than one replicated service thus reducing the task of network-wide service assignment in Jini federation to the level of local (intra-node) task scheduling. I.e. the constancy of number of nodes limits somehow the dynamics of the number of services.

The cycle of interconnection steps between the Jini components and protocols is outlined in Fig. 1. According to the Jini architecture the system comprises at least one lookup service l (or $l_1, l_2, ..., l_L$) holding the registration of all other services in the Jini community. A service exists for the federation only after the processes of lookup discovery (step 1) and registration with the lookup service using Jini's join protocol (step 2) have completed.

The clients are the dynamic component of the system architecture. They are processes, which access the system resources in two steps: first they receive the proxy object of the assigned service in the lookup boot process (step 3)

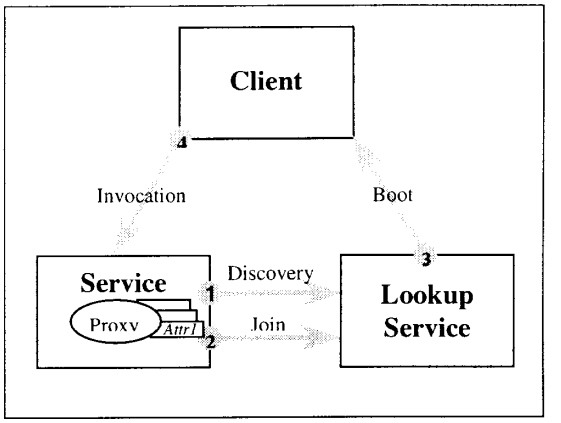

Figure 1. The Jini process of assigning a service to the client.

and then they perform the service invocation (step 4). A client locates one of the replicated services using the lookup protocol. As the services are replicated the clients compete for the same resources. The local scheduling of the clients' lookup requests at the lookup service follows the FCFS (first-come-first-served) discipline. Once served the clients leave the system.

In general the process of arrivals of clients in the system is probabilistic. We consider two arrival scenarios:

- Independent individual clients. The clients arrive one at a time separated by independent arbitrary intervals. Interarrival independence means that the probabilistic arrival process is a Poisson process. This case is applicable by most of the user-initiated calls and a substantial part of machine-initiated calls to the Jini services;
- Group clients. The clients arrive in a group forming a process known as bulk arrival. Typically, this scenario takes place when a big application has been decomposed by the user to a group of small sub-tasks - the actual clients, any of which has to be processed by one of the replicated services in Jini. As with the replicated services, the group clients though analogous are not necessarily equivalent in a performance sense (i.e. they may require different service times).

Clearly, these two scenarios do not cover all possibilities for the client arrival process in an arbitrary Jini federation. One may find systems where clients arrive in some deterministic order or where the interarrival times are probabilistic but not independent. (Probably the most significant case of the last option is the web page operations). Another topic of consideration is whether the clients are associated with a scale of service priorities. The present study does

not address such eventualities.

The system architecture is obviously that of a network-based distributed system. Yet logically it is lookup-centric, which challenges the possibility for a bottleneck at the lookup service. The obvious alternatives to the bottlenecked federation-unique lookup service are either splitting the clients, splitting the scope, or doing both:

- Lookup matching. The actual population of clients is split between L lookup services. A client is associated arbitrarily to one of the lookup services upon its arrival in the system. The association is arbitrary for the client; it might be RR (Round Robin), JSQ (Join Shortest Queue), etc. for the lookup service. Each lookup service acts to its clients as if it is the only lookup service in the federation.

- Lookup decomposition. A Jini community of services is split among the available L lookup services. The lookup scope of each service is reduced by the factor of L. While not favouring the best possible balancing among the services this measure can be found handy in the case of expanding the network scale.

- Multi lookup. Both the service scope and the clients are split among the available lookup services.

All three approaches are entirely legitimate in Jini. Lookup decomposition seems more sparing in what concerns the overload due to discovery/join/lookup protocols trinity. Lookup matching preserves the system-wide scope of replicated services, which might be advantageous in a performance sense. We call these three variants of multiple lookup services hierarchical lookup because the lookup process is actually two-fold: client-lookup service association and then client-service association.

3. Service assignment schemes in Jini

In our comparative analysis and development of different service assignment schemes the services are "receivers" of tasks and the clients are "senders" of the tasks. This is to justify that we do not consider the possibility for inter-service balancing transfer of tasks - from "sender"-service to a "receiver"-service. Although perfectly feasible and legitimate such a technique falls somewhat out of the scope of Jini control tools. We merely treat all services as "receivers" of tasks and consider the decision taking instants (apparently by the lookup service) recognising two basic available options - assignment upon arrival of the clients and assignment upon the re-emerging of a replicated service (i.e. upon the departure of its serviced client). Furthermore, for the standard assignment upon client arrivals we study the performance of a four level information policy. All these options form several service assignment schemes. Our results demonstrate the performance benefits of the proposed delayed assignment as well as the limitations of the system granularity and scalability.

In a lookup-centric Jini federation the service assignment task is performed by the lookup service. This architectural prerequisite favours any load balancing scheme that places information and location strategy together with the lookup service. The functionality of the transfer strategy is virtually not presented in terms of Jini technology. This is due to the fact that once established, the client-service assignment is supposed to last during the lifetime of the client object. Presumably, it is not the service's task to form complementary pairs and to swap the most delayed clients. At the same time nothing prevents the system from redistributing dynamically the clients already in service (or waiting for service in the local queues) according to algorithms in [3, 12, 14].

The core of the information policy is its scale. The information scale of a Jini lookup service may have four levels, namely:

- Level 0 (`L0`): no information about services; one state of the service - "available".

- Level 1 (`L1`): reference of the local service queue lengths q_i to given threshold $T(q_i < or \geq T; i = 1,..S; T \geq 1)$ - two states of the service: "underloaded" `L1-` and "overloaded" `L1+`. Note that for $T = 1$ `L1` coincides with `L0` reducing the scale under-/over-loaded to idle/busy.

- Level 2 (`L2`): works like `L1` for $T \geq 2$ plus information about the idle services; three states of the service: "idle" `L2o`, "underloaded" `L2-` and "overloaded" `L2+`.

- Level 3 (`L3`): full actual information about the local queue lengths of the services. The service's states are boundless (or at least numerous).

From a performance point of view, the most valuable information is whether the service is idle (or about to be idle soon), or not [3, 10]. That is why `L1` or `L2` seem more promising than `L3`.

The next key issue concerning the assignment scheme is the assignment initiative, which is the substance of the location strategy. In Jini federation we apply three types of assignment:

- Early assignment `!A` (clients' initiative).

- Delayed assignment `A!` (service's initiative).

- Hybrid assignment `!A!` (mixed initiative)

In the case of early assignment (we use the short notation `!A`) the binding of the client-service pairs takes place upon arrival of every new client in the system. Ideally, there is no queue of waiting clients in front of the lookup service. The queuing is localised in front of the services. The location decision itself is taken by the lookup service based on the available information. For `L0` obviously the clients will be assigned to the available services arbitrarily (technically, the lookup service may perform RR, which can give slightly different results to these of a proper random assignment Rnd). Note that the combination `L0_!A` (i.e. no information gathering with an arbitrary early assignment among the available replicated services) is the "standard" assignment scheme in Jini federation. This standard scheme is applicable whenever no assignment consideration has taken place on the stage of system coding. Early assignment can also be combined with the other three information levels producing schemes such as Join Idle/Underloaded/Shortest Queue (JIQ/JUQ/JSQ, which correspond to `L1_!A`, `L2_!A` and `L3_!A` respectively).

More promising to us seems the delayed assignment (noted here with `A!`). It corresponds to the service's initiative. In this case the assignment takes place whenever a service becomes idle. This means that arriving clients are forced to join a common federation-wide queue and wait until there is an idle service to which a waiting client is to be assigned. The advantage of this discipline is that the set of local queues is replaced by a common federation queue. In queuing theory this case is classic. The common queue (a true invention of 1960s) outperforms the multiple separate queues and often is referred to as "fast" queue [8]. The delayed assignment goes with `L0`.

By hybrid assignment (noted `!A!`) queues are formed to all the services including the lookup service. Assignment may take place both at the time of a client's arrival or on demand of the services. Every client may experience either early assignment or delayed assignment depending on the current conditions of the system. Here all information levels are applicable. In practice, the presence of a federation queue (or "fast" queue) allows the usage of some limited in size local queues. Their function is not to be that of a principal distributed repository of the waiting clients' pool but merely that of compact local buffers, which make the events of transition between the states of the services more rare (e.g. underloaded-overloaded instead of idle-busy), thus potentially reducing the system control overhead. On the other hand long local queues are to be discouraged as even the most sophisticated information and location strategy `L3_!A` (JSQ) is worthless if the choice is between any two services with perhaps 30 and 33 clients waiting in their local queues (provided the service times form a probabilistic sequence).

The three assignment schemes - early, delayed and hybrid - are also applicable in the case of distributed processing of a "big" decomposed task. If the system comprises of N nodes running S (here $S = N$) replicated services, then distributed processing of that task requires decomposition of the task's domain into S or more subtasks which are assigned at the same time to the separate services. Such an approach belongs to the static load balancing techniques (i.e. to early assignment `!A` in our taxonomy). A significant performance improvement can be obtained by application of so called scattered decomposition, which distributes the task into $kS(k > 1)$ subtasks and assigns them in an arbitrary order to the different services [5]. Delayed assignment `A!` of such decomposed tasks would mean that after initial assignment of first S subtasks to the available replicated services, the rest $(k-1)S$ subtasks are queued and assigned along with the availability of idle services in the Jini community. This scheme can also be hybridised by allowing some short limited local queues to the services.

All these considerations can be extended for a Jini federation applying hierarchical assignment, i.e. comprising more than just one lookup service. Then the multiple lookup services are to be treated in the same way as the services are in the above paragraphs.

4. System models

Following the structure of a Jini federation, our model consists of persistent components and transient components or *transactions*. The last ones are the clients. Every client is identified by two temporal quantities: the moment of its arrival in the system and the service time it requires. Although seeking the same service the clients are not necessarily allocated the same service time. In fact most of the numerical tasks as well as the communication or peripheral devices tasks take unknown time to be completed. That is why the service time parameter is not to be used on the stage of service assignment but only when the transaction enters the service stage of the model. In heterogeneous systems the service time depends on the allocation as there are "faster" and "slower" nodes. For such a case the service time has to be considered as basic service time subject of correction by appropriate coefficients with regard to the actual allocation. Our model recognises two types of transaction arrivals: continuos flow of transactions with exponentially distributed interarrival times and bulk arrival of transactions, when the important parameter summarises as the number of transactions instead of the time of their arrival. The concept of the transactions can be extended with one more numerical component - priority. In this paper we do not consider priority clients. After assignment and completing the service the clients-transactions leave the system. Their mean lifetime or the duration of their residence cycle

in the model is the main criteria for the effectiveness of the modelled system.

Usually the persistent model components are called *blocks* [13]. There are two types of blocks in our model: queues and devices. They represent the services and the lookup service itself (which is more than one in case of hierarchical assignment).

The early assignment !A means that the transaction queues are placed in front of each service. Thus services appear to be queued devices. The lookup service is just an instantly working device, which splits the flow of arriving transactions according to the practised information level. The length of the services' queues must prevent discarding of transactions because of lack of buffer space. The graphical presentation of !A-blocks is given in Fig. 2a.

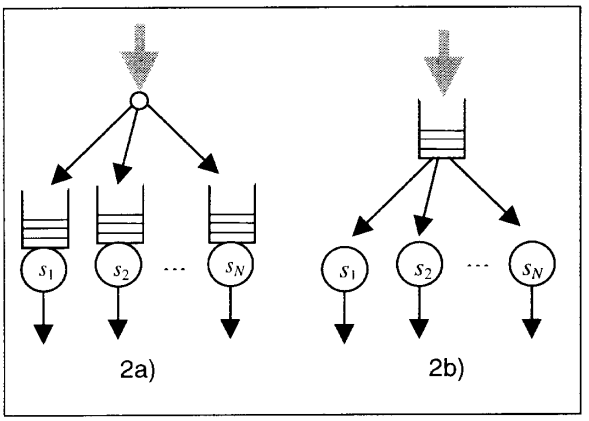

Figure 2. Assignment modelling: 2a) Early assignment scheme; 2b) Delayed assignment scheme.

By delayed assignment A! the lookup service is modelled by a common waiting line and an instantly or promptly working splitting device. The services are queueless devices having two basic states - idle and busy (Fig. 2b). This model can be detailed by splitting further the busy state into three states - leave, busy and join, during which the service is not available for assignment to the clients. Such a diagram represents more adequately an implementation of the Jini services leaving the federation while busy and preventing thus further allocations of clients during the busy period and the leave/join periods framing it.

The hybrid assignment !A! implies the most complicated state-transition diagrams for the blocks. The lookup service is modelled like for early assignment !A. However unlike !A, here the services are queued as well. In this model, the availability of the common fast queue allows the local service queues to be limited in size without the risk of loss of transactions (clients). Although not necessarily outperforming simple A! schemes, the hybrid schemes may prove (based on results in [3]) to be effective in reducing the control overhead thus securing a better scalability and finer granularity (when sought) of the Jini based distributed processing.

Along with the models the following parameters are introduced. Mean residence time T spend by each client in service. T excludes the time for assignment i.e. the time of processing by the lookup service, which is noted τ. Thus the total mean time spend by the client in the system equals $T + \tau$.

Performance rate of the lookup service to process any of the operations "join", "leave" and "assign service" is μ. For simplicity (but not necessity) we assume that the tree operations last approximately equally. In the case of continuos arrivals of transactions the value of the relative system load $\rho = \lambda/(S\alpha)$, where λ is the transactions'/clients' arrival rate and α is the processing rate of each of the replicated services. By a heterogeneous system the multiplication in denominator has to be naturally replaced by the sum of the processing rates.

We introduce a specific quantitative parameter to measure the system granularity γ. It is difficult at this level of abstraction to compare the clients' complexity in machine instructions, so we compare the last with the complexity of join/leave/assign protocol operations, which doubtless are all fine-grain processes. So the granularity in our model is measured by $\gamma = \mu/\alpha$. Finest granularity corresponds to small values between 1 and 10-20 times the complexity of protocol operations. The upper boundary of γ, which corresponds to coarse-grain client tasks, is not limited.

5. Case study of level L0

The service assignment models can be studied by means of simulation programming and benchmarking. The simplest of the schemes however are subject of formal estimation as well. Here we employ a formal case study of the L0 information level for several reasons. These results are of practical importance, as L0 appears to be most popular and heavily employed in Jini distributed applications. Indeed it includes the simple "standard" association L0_!A. Nevertheless, it presents a good opportunity to compare early to delayed assignment schemes (to which latter we advocate though not unconditionally) and to extend the study to the issues of scalability and granularity. Furthermore, the formulae are more useful than the simulation or benchmarking programs when a brief estimation of the system performance is sought (subject of fairness of the values replacing the system parameters).

If the interarrival and service time of the transactions are both distributed exponentially, and Rnd (or RR) is applied as location strategy, then for L0_!A (the standard service

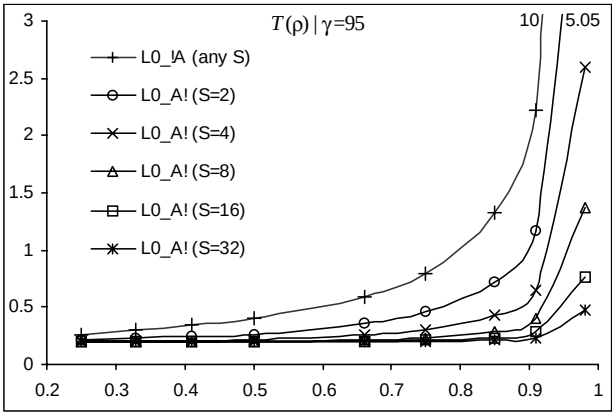

Figure 3. Mean residence time in service by different system load. Delayed vs. early assignment.

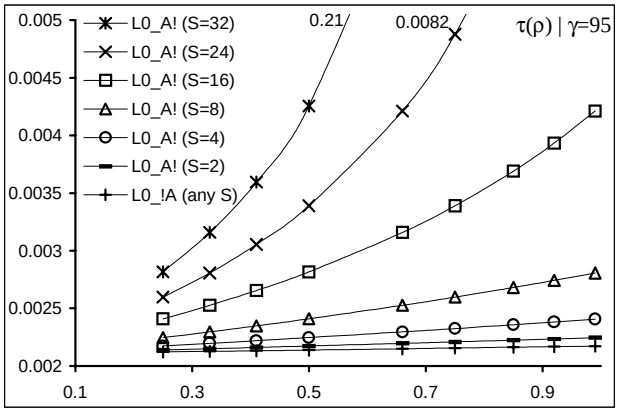

Figure 4. Mean time for assignment by different system load.

assignment in Jini federation) we have a case of a M/M/1 queuing system for each of the services and the calculation of the mean residence time of the clients/transactions in the system is straightforward. If the system is heterogeneous (i.e. the mean service time of the replicated services differs) than an iterative estimation of the probability of assignment will yield the steady state results for all the services, which to be summarised statistically (in manner this has been done in [3]).

A natural counterpart of the standard assignment is `L0_A!` or `L1(T=1)_A!` (delayed assignment with arbitrary choice of available idle services). The homogenous case of it represents simply a M/M/S system (where S is the number of all services in Jini community. Using Erlang's result from [8] we obtain the mean residence time for such a system by different relative load and different relative granularity in the system. The results presented in Fig 3. prove the effect of application of delayed association to the early one. Technically this requires replacement of the [transparent] local queues of the services by a common "fast" queue for the lookup service (or services).

Although promising as they are the above results are valid if one ignores the overhead imposed by supporting the lookup service's queue. However such a centralised scheme may account for a bottleneck in the distributed systems. To correct the model we have to consider the delay due to service-client assignment (which is inevitable) as well as that to join/leave protocols by the services[1].

[1]One may apply mechanisms other than simply using the join/leave protocols in order to obtain distraction of the newly assigned services from the Jini community. In any case, they will impose some delay in servicing the clients along with loading the lookup service with that additional protocol overhead.

Presumably the rate at which the lookup service (when idle) performs join and leave operations is known - μ. Considering lookup service as single server queuing system, we have to define its arrival process which does not coincide with the clients' arrival process. In `A!` case the operation of the lookup service is initiated by the services instead of by the clients. Thus the lookup arrival process has rate φ such that $\varphi = 3S\alpha\rho = 3S\mu\rho/\gamma$ (from the queuing theory $\alpha\rho$ is the rate of departure of serviced clients which correspond to a new join process for the now idle service; this value is multiplied by the number of services S and by the number of lookup operations - join, leave and assign). The mean time for assignment is $\tau(\varphi, \mu)$ and it incorporates all tree protocol operations as they rise with the same frequency, though in different moments.

Considering the protocol overhead of the delayed assignment, we calculated the time τ spent in service-client assignments for different system load ρ and fixed (fine to moderate) system granularity $\gamma = 95$ (see Fig. 4). Here we observe that the better performance of the delayed assignment schemes regarding T is countered by lesser stability regarding τ. Indeed, the stability of the lookup process by "standard" (i.e. early) assignment is confirmed by the experimental results in [7]. This implies the probable restrictions to the system scalability of the delayed assignment protocols.

Another subject of interest in our case study is the impact of the system granularity, which is presented here in Fig 5. The results show that the delayed assignment is applicable for systems of medium and coarse granularity but has questionable merits for fine granularity of the clients' tasks. The missing points of the `A!` graphs correspond to situations for which there isn't a steady-state solution.

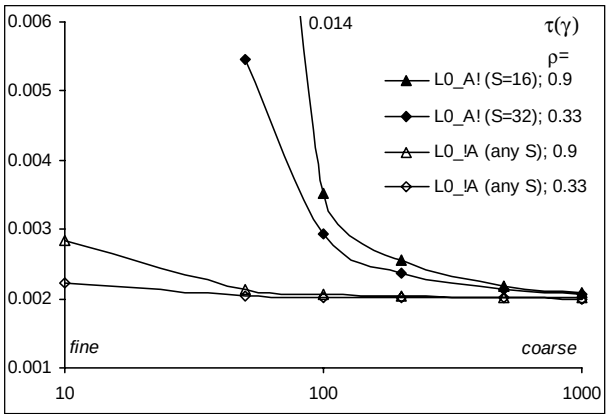

Figure 5. Mean time for assignment by different system granularity.

6. Conclusions

The Jini architecture is focused on putting the system together by higher-level structuring and linking the clients and system resources. The key QoS issues of the Jini technology are simplicity, reliability and spontaneity. This paper suggests a set of schemes that would bring additionally to the structure of Jini components the benefits of better usage of the potentially available multiple resources in the network. While a "standard" Jini program would employ an assignment scheme, which is practically an early assignment without local system information (`L0_!A`), it is still worth to consider other options such as delayed assignment or assignment based on some more details about the dynamic load of the services in a federation.

Our case study demonstrates the advantages and limitations that the delayed service assignment may have in a Jini-based distributed processing system. Yet, it is merely an example of how the presented assignment schemes can be evaluated and what sort of results one should expect. Obviously, it is only a fraction of work due to the performance evaluation of distributed Jini applications. Future research in this direction would include simulation and benchmarking studies of the performance effect for different information levels. The important cases of both early and delayed assignments of a bulk arriving decomposed application has also to be evaluated. We expect that extensive implementation hints will emerge when the whole picture of performance evaluation and applicability of the assignment schemes is being completed. Indeed, such type of results can be used for the design of dynamically adjustable service assignment schemes.

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. *Operating Systems Review*, 34(5):186–201, December 1999.

[2] K. Arnold. The Jini architecture: Dynamic services in a flexible network. In *Proc. of 36th ACM/IEEE Design Automation Conference*, pages 157–162. ACM Press, June 1999.

[3] V. Georgiev and M. Iliev. Hybrid scheme for load balancing in heterogeneous distributed systems. In *Proc. of 11th European Simulation Multiconference*, pages 521–528. SCS, June 1997.

[4] V. Getov, P. Gray, S. Mintchev, and V. Sunderam. Multilanguage programming environments for high-performance Java computing. *Scientific Programming*, 7(2):139–146, April 1999.

[5] R. v. Hanxleden and L. R. Scott. Load balancing on message passing architectures. *Journal of Parallel and Distributed Computing*, 13(3):312–324, November 1991.

[6] P. Hasselmeyer, M. Schumacher, and M. Voss. Pay as you go - associating costs with Jini leases. In *Proc. of 4th International IEEE Conference on Enterprise Distributed Object Computing (EDOC'2000)*, pages 48–57. IEEE Computer Society, September 2000.

[7] M. Kahn and C. Cicalese. CoABS grid scalability experiments. In *Proc. of Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, May 2001.

[8] L. Kleinrock and R. Gail. *Queueing Systems: Problems and Solutions*. John Wiley & Sons, New York, 1996.

[9] F. Kon, R. Campbell, M. Mickunas, K. Nahrstedt, and F. Ballesteros. 2K: A distributed operating system for dynamic heterogeneous environments. In *Proc. of 9th IEEE International Symposium on High-performance Distributed Computing*, pages 201–208. IEEE Computer Society, August 2000.

[10] H. Kuchen and A. Wagener. Comparison of dynamic load balancing strategies. TR 90-05, RWTH - Aachen, 1990. ftp://ftp.informatik.rwth-aachen.de/pub/reports/1990/index.html.

[11] Q. Mahmoud. Using Jini for high-performance network computing. In *Proc. of IEEE International Conference on Parallel Computing in Electrical Engineering (PARELEC'00)*, pages 244–247. IEEE Computer Society, August 2000.

[12] R. Mirchandaney, D. Towsley, and J. Stancovic. Adaptive load sharing in heterogeneous distributed systems. *Journal of Parallel and Distributed Computing*, 9(4):331–346, August 1990.

[13] T. Schriber. *An Introduction to Simulation Using GPSS/H*. John Wiley & Sons, New York, 1991.

[14] N. Shivaratri and P. Krueger. Adaptive location policies for global scheduling. *IEEE Transactions on Software Engineering*, 20(6):432–444, June 1994.

[15] J. Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, July 1999.