
In this chapter we describe how we implemented a simulator to verify hypothesis made in previous chapter about multicast injection for resource-initiated allocation deployment mechanisms. Afterwards we show simulations conducted and results obtained.

As it was commented in previous chapter we could not verify hypothesis with a prototype implementation because we did not have enough resources to conduct such large-scale experiments. Another method to evaluate some mechanisms is analytical modelling, however for such complex scenarios as the Internet, so many abstractions need to be made that usually results have little significance. The third method, simulation, permits to evaluate fairly complex system with affordable computing resources, and to reproduce experiments multiple times under equal conditions.
6.1. Simulation Methodology

We simulated both resource allocation mechanisms and their respective deployment mechanisms using the ns-2 network simulator [McC98]. It is a simulator widely used in network research, very stable, with source code available. Besides for these particular simulations there are three technical reasons for choosing it:

- It allows configuring network topology, link bandwidth and latency.
- Multicast support is implemented; therefore services using it, such as multicast injection, are easily implemented.
- There are application layer modules that can extended to implement deployment managers and resource agents.

6.1.1. Topology, Resources and Deployers

We configured a wide area network with 5 top level regions, each region has an exchange point connected to 2 international exchange points by 1 Mbit/50ms delay links, each of the 5 international exchange points is connected to 2 other international exchange points by 1Mbit/100ms delay links. There are 25 edge regions, 5 connected to each top level regions by 4 Mbits/20ms delay links. Figure 2 shows this scenario.

There are 25 resources nodes, one at each edge region. Each resource node has 1000 Mbytes storage and 10 Mbits service traffic capacity, typical values of Internet hosting servers. Each node has 1 edge region at distance 1, 4 edge regions at distance 3, 10 edge regions at distance 5, and 10 edge regions at distance 6.

And there are 5 deployers, each connected to one top level regions by 4 Mbits/20ms delay links.
6.1.2. Application Network Specification Parameters

Deployer entities accept deployment requests with these application network specification parameters:

- Per node storage, required storage capacity at nodes for application data and code.
- Network traffic, represents expected network traffic capacity required for client requests to application. It can be expressed as traffic-region pairs. In this simulation expected traffic is equal at all regions.
- Network regions, where service is to be provided, currently we are considering it is a list of AS numbers.
- Maximum distance, between service nodes and client regions, represents required service level. Selecting larger distances means worse services are tolerated. Currently, distance is the number of Internet hops between client regions and nodes.
• Number of nodes providing service, if a large number is specified, the service will have greater fault tolerance and availability, however it represents a higher cost.

• Service duration, it will express a determination to make use of resources for a specified period, which can be changed later.

On table 6.1. we show which parameter values were choosen for simulation. Every application network to be deployed is specified to request 100 Mbytes storage and 1 Mbits traffic capacity per region, which represent 20 simultaneous clients connected by dial up modems (these values have been chosen so that every resource node has exactly 10 resource units, being one resource unit 1 Mbit traffic capacity and 100 Mbytes storage capacity); service is required at 5 random regions, all with equal probability; service has to be provided from 5 nodes; maximum distance between surrogate and client is 1 hop, therefore each service can only be serviced by those five nodes at each requested region. And service lifetime has a negative exponential distribution with an average of 100 seconds. It is a low value for normal Internet service life times, though if we consider this value to represent interval of modification of service demand, it is fairly representative of many popular flash-crowd services [Ren02].

<table>
<thead>
<tr>
<th>Application Network Specifications</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>100 Mbytes per node</td>
</tr>
<tr>
<td>Network Capacity</td>
<td>1 Mbits per region</td>
</tr>
<tr>
<td>Regions</td>
<td>5, random</td>
</tr>
<tr>
<td>Distance</td>
<td>1 hop</td>
</tr>
<tr>
<td>Number Nodes</td>
<td>5</td>
</tr>
<tr>
<td>Service Duration</td>
<td>Neg. Exp. Distribution:</td>
</tr>
<tr>
<td></td>
<td>avg. 100 sec.</td>
</tr>
</tbody>
</table>

*Table 6.1. Application Network Specifications Values*
6.1.3. Agents and Allocation Protocol Implementation

Resource agents and deployment managers have been implemented extending ns-2 Process and Agent classes for unicast and multicast based communications respectively. Unicast communications are implemented using TCP implemented in FullTCP class we tried to use SimpleTCP class; which intends to provide similar behaviour with much less computational requirements, however it was not fully implemented. To implement request events and resource pools AppPool class has been used. It has been made some simplifications to deployment mechanisms described at previous chapter, since we were interested in studying its resource allocation behavior.

Deployment mechanisms implementing a service-initiated allocation do not implement resource discovery, resources nodes are configured at the initialization script. Resource availability is maintained at each deployer in a table, which monitors every resource by periodically sending unicast monitoring messages. Monitoring rate is configurable: experiments have been carried out with monitoring rates of 5 and 30 sec. For each deployment request, resource mapping algorithm shown in fig. 5.7. is executed and allocation commands are sent to selected nodes. If every allocation is successful, deployment is successful. If some allocation is not successful due to contention, such deployment will be cancelled after a timeout, which has been set to 3 seconds. Deployment requests that cannot map requirements into deployment managers resource availability table will be rejected, however in some cases such information will be stale and deployment might have been successful. After the requested service duration successful deployment will be torn down.

Multicast injection for resource-initiated allocation of application network deployment implements all phases shown at previous chapter, except multicast damping. Every deployment request was configure to request resources from 5 regions at a distance of only one hop away, since at each region there exist only one resource node it is not possible to
overprovision any service. Therefore there is no need to implement multicast damping, and we can evaluate the effect of autonomous resource allocations isolated. Resource agents listen to multicast injection messages; if they have requested resources and can provide service to the regions indicated at the service level required, they allocate resources and publish it in a service specific multicast channel that has been indicated in the request. Deployers inject all specifications; they do not reject any deployment request. When deployers listen that all required resources for a services have been allocated, they consider it as successful and start a timer to inject a service termination message when its duration is reached. If after a timeout deployers do not listen enough resources having been allocated, they cancel deployment and send a multicast release message. Such time out have been configure in 3 seconds.

<table>
<thead>
<tr>
<th>Centralized Service–initiated Allocation Algorithm</th>
<th>Multicast Injection Resource–initiated Allocation Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each_Deployer {</td>
<td>Each_Deployer {</td>
</tr>
<tr>
<td>:: deployment_request {</td>
<td>::deployment_request {</td>
</tr>
<tr>
<td>select_nodes();</td>
<td>multicast_injection();</td>
</tr>
<tr>
<td>If enough nodes selected</td>
<td>}</td>
</tr>
<tr>
<td>{</td>
<td>}</td>
</tr>
<tr>
<td>Foreach selected node</td>
<td>All_Resource_Agent {</td>
</tr>
<tr>
<td>{</td>
<td>::receive_mcast_injection {</td>
</tr>
<tr>
<td>command_allocation();</td>
<td>If enough resources &amp;&amp;</td>
</tr>
<tr>
<td>}</td>
<td>at indicated distance of</td>
</tr>
<tr>
<td>}</td>
<td>some service regions</td>
</tr>
<tr>
<td>:: select_nodes {</td>
<td>}</td>
</tr>
<tr>
<td>Foreach service region</td>
<td>allocate();</td>
</tr>
<tr>
<td>{</td>
<td>mcast_allocation_published();</td>
</tr>
<tr>
<td>among those with</td>
<td>}</td>
</tr>
<tr>
<td>enough resources,</td>
<td>}</td>
</tr>
<tr>
<td>select resource</td>
<td>}</td>
</tr>
<tr>
<td>at indicated distance;}</td>
<td>}</td>
</tr>
</tbody>
</table>

*Figure 6.2. Allocation Algorithms in Simulation*
6.1.3. Choice of deployment request rate and resource load parameters values

Deployment request rate values were chosen because we were interested in observing behaviour of deployment mechanisms when resources are scarce. Therefore we have to find out which deployment request arrival rates “saturate” such resource pool. We obtained this value with following method. Little’s law [Lit61], which is represented as:

\[ E(N) = \lambda * E(T) \]  \hspace{1cm} (Eq. 6.1.)

It states that the average number of entities in a system \( E(N) \), is equal to its arrival rate \( \lambda \), multiplied by its average service time \( E(T) \). Therefore if a system has only capacity for \( M \) entities, a service input whose arrival rate by average service time is over \( M \), is bound to saturate the system and have many service inputs rejected.

Applying such method to our case, we first need to obtain maximum number of application networks that can be deployed at resources being provided. There are 25 resources nodes each with 1000 Mbytes and 10 Mbits, and each deployment request demands 100 Mbytes and 1 Mbits at 5 random different nodes, therefore there is an upper limit on the number of simultaneous applications which can be deployed:

\[
\text{Max. Deployed Applications} = \frac{\text{Num. resource nodes} \times \text{Resource units per node}}{\text{Resource units per application}} = \frac{25 \times 10 / 5}{50} \]

This is a very high limit, and not very probable of being reached. It will not be very probable that all resources at all nodes are allocated simultaneously because nodes are chosen randomly by deployment entities. Therefore once very few resources are not
allocated, the probability of being chosen by deployers is low, unless deployers do not care about its characteristics are willing to choose resources with worst properties, such higher distance to clients, etc.

Application networks have a duration that follows a negation exponential distribution with 100 seconds average time. Therefore applying Little’s formula the aggregate deployment request rate that fully occupies the system is:

\[ \lambda = \frac{E(N)}{E(T)} = 50 / 100 = 0.5 \text{ requests per second} \]  

(Eq. 6.3.)

Therefore we made experiments varying aggregate deployment request rate from 0.1 to 0.6 requests per second, which corresponds to per deployer average arrival rates varying from 1 request every 50 seconds to 1 request every 8 seconds.

A better metric to measure performance of resource allocation is resource load, which indicates how many resources are being demanded. Each deployment request demands 100 Mbytes and 1 Mbits at 5 nodes, that is 5 resource units. Because there are from 0.1 to 0.6 deployment requests per second, resource load varies from 0.5 to 3 resource unit requests per second.

<table>
<thead>
<tr>
<th>Demand</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution type</td>
<td>Poisson arrival rate</td>
</tr>
<tr>
<td>Average Request Rate</td>
<td>1 / 50 – 1 / 8 req. per sec.</td>
</tr>
<tr>
<td>Aggregate Rate</td>
<td>0.1 – 0.6 req. per sec.</td>
</tr>
<tr>
<td>Resource Load</td>
<td>0.5 – 3 resource units / sec.</td>
</tr>
</tbody>
</table>

*Table 6.2. Simulation Parameters*
6.2. Measurements

From hypothesis set at the end of previous chapter we can derive evaluation criteria, and measurements to be made. Our first metric is number of successful deployment requests. Other metrics for comparison are success deployment ratio and traffic created. To verify some hypothesis resource contention and unused resources allocated need to be measured.

**Successful deployment requests:** each application that has been allocated request resources is consider successfully deployed. A deployment request is not successful if a) a deployer rejects it because it considers there will not be enough resources based on its resource availability information, or b) some allocation required have not been made, after a timeout such allocations have to be released by a release request.

**Success deployment ratio:** metric can be derived from previous metric simply by dividing it by total number of requests. It depends on various factors: number of deployers, the more deployers more contention for resources and higher probability of failure; service specifications, stricter specifications means an application has higher probability of failing to be deployed.

**Resource contention:** Centralized allocation deployment mechanisms suffer from resource contention. Deployers with stale resource availability information try to allocate resources that have already been allocated by others deployers. Resource contention probability increases with resource load, and number of deployer entities. It can be measure by number of deployment requests which initiated resource allocation but failed.

**Unused allocations:** occur whenever a resource is allocated but does not provide a service because such allocation is cancelled before service is activated. It can be a problem if they represent a large percentage of resource occupancy time. This phenomenon has been called thrashing, and can lead to blocking.
**Bandwidth:** We want to measure bandwidth required at every deployer’s location, and total bandwidth consumed on the network. It is a function of deployment request rate and monitoring request rate, number of resource nodes and deployer nodes and message sizes.

<table>
<thead>
<tr>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful Deployments</td>
</tr>
<tr>
<td>Success Ratio</td>
</tr>
<tr>
<td>Failed deployment requests</td>
</tr>
<tr>
<td>Unused Allocations</td>
</tr>
<tr>
<td>Total Network Bandwidth</td>
</tr>
<tr>
<td>Deployer Bandwidth</td>
</tr>
</tbody>
</table>

*Table 6.2. Simulation Measurements*
6.3. Simulation Results

Simulated time was 2000 seconds, leaving 200 seconds for warm-up before start collecting data. Simulations were performed on a SunOS machine with 256 Mbytes of RAM. Simulation time varied from 15 minutes to 5 hours.

6.3.1. Impact of Resource Load

We obtained number of successfully deployed applications and success ratio for different resource loads. In figures 6.4. number of successfully deployed applications increases as resource load increases for multicast injection mechanisms. However for service-iniciated allocation deployment mechanisms, number of success deployment application decreases as resource load increases beyond 2 resource units requested per second.

![Figure 6.4. Successful Deployments vs. Resource Load](image)

In figure 6.5. success ratio keeps high for moderate resource loads and decreases for high loads for both mechanisms. Because we are allocating resources only when they are one
hop away from the region requested, the rate of successfully deployed applications decreases very fast when resource load increases. If we allowed applications to be allocated in farther nodes, success ratio will be a bit higher, however at the cost of providing worst services.

Centralized service provider commanded allocation deployment has worst behaviour than multicast injection at high loads, reaching a low 30% of success ratio, and even decreasing total number of successful deployments because it makes use of stale information. Stale resource availability information in service-inciated allocation deployment causes deployer entities to select nodes that have been allocated, and not to consider for allocation nodes that have already available resources. Deployers consider many resources as fully allocated and reject deployment requests. In figure 6.6. we show number of application deployment requests which mapped into a set of resources at deployment managers, but failed deployment due to contention among different deployers. Resource contention increases with resource load, causing total number succesful deployments to decrease with resource load.

Figure 6.5. Success Ratio vs. Resource Load
With multicast injection no deployment request is rejected by deployers, all requests are injected and resources allocate autonomously. Success rate is higher as seen in previous figure 6.5. However, as it has been discussed, with this mechanism a number of allocations are made which are never used by an application, because not enough resources were found. These can lead to undesired thrashing. Next we study these phenomena.

In figure 6.7. unused allocations, as percentage of total possible allocations, has been plot against resource load. Unused allocations occurs whenever resources are allocated and released shortly afterwards without providing any service in that period. Multicast injection shows high unused allocations values, however they represent only a very small percentage of total resources. It seems to increase exponentially as deployment request load increases, however at highest simulated resource loads, it only represents a 3.3 % of total resources. This is so because service lifetimes (media 100 secs.) are much higher that cancellation timeouts (3 secs.), therefore for a given request load most of the time resources are servicing applications, and cancellations consume very few resources. Centralized service-initiated allocation also
exhibits this phenomenon, though in an smaller scale. Allocations corresponding to deployment requests that cannot allocate some resources due to contention have to be released without having been used.

Adequate resource release timeout values are require so that unused allocations represent only an small percentage of total resources occupancy. Since unused allocations are proportional to resource load, another recommended mechanisms to avoid a high percentage of unused allocations are input limiting filters [], which limit resource load at each deployer.

![Figure 6.7. Unused Allocations vs. Resource Load](image_url)

Last measurements made are total network bandwidth and bandwidth at deployers locations. In figures 6.7. and 6.8. we present bandwidth required by each type of mechanisms on every deployers location and total network bandwidth required at all the network by all deployers as the sum over all links.

Centralized service-iniciated allocation deployment requires much more bandwidth at deployer locations than multicast injection. Multicast injection generates all of its traffic by multicasting service specifications requests and cancellation request. As expected, it grows with increased deployment request rate, but it is a very low value at deployer locations.
Centralized service-initiated allocation deployment generates most of traffic to monitor status of each resource node. This traffic does not increase as resource load goes up. There is a small component of traffic created due to resource allocation request messages, which increases slowly as deployment request rate increases, until resource load is so high that many requests are rejected by deployers without contributing more allocation messages.

**Figure 6.8. Per Deployer Bandwidth**

Concerning total network bandwidth required, multicast injection produces a large amount of network traffic on the whole network that increases linearly as deployment requests rate increases. A means to diminish it would be to distribute multicast injection requests into several multicast deployment channels created with AWC group formation protocol [Zha98], which provides mechanisms so that different multicast groups are created among a large number of components.

Centralized resource allocation requires a total network bandwidth that is proportional to deployers’ bandwidth as seen from similar figure pattern.
6.3.2. Impact of Monitoring Rate

When monitoring rate is increased from 30 seconds to 5 seconds number of successfully deployed applications with centralized resource allocation increases, fig. 5.16, due to less resource contention fig. 5.17, however at the cost of higher deployers bandwidth, fig. 5.17, and total network bandwidth, fig. 5.18.
### Figure 6.10. Success Deployed Applications

- **MI**
- **CC (Mon. rate = 30 sec.)**
- **CC (mon. rate = 5 sec.)**
- **Polinómica (CC (mon. rate = 5 sec.).**

### Figure 6.11. Resource Contention for Centralized Allocation

- **CC (mon. rate = 30 sec.)**
- **CC (mon. rate = 5 sec.)**
Figure 6.12. Per Deployer Bandwidth

Figure 6.13. Total Network Bandwidth
6.3.3. Impact of Number of Deployers

We have run the same simulation varying the number of deployer entities from 5, 10 and 20. To keep total resource load constant we have decreased each deployer request rate proportionally by 1, 2 and 4.

In figure 6.18. and 6.19. we see how many application networks are successfully deployed as the number of deployment entities changes from 5 to 20 at different deployment loads. With a moderate resource load of 1.5 resource unit requested per second, fig. 6.18., every deployment mechanisms provides a constant success ratio as the number of deployment entities increases. However when we increase total request load to 2.5 resource units requested per second, centralized resource allocation deployment mechanisms decreases its success ratio as the number of deployment entities increases. Again, it is worst for low monitoring rates. It occurs due to increase resource contention with increased number of deployment entities.

In figure 6.20. centralized resource allocation deployment increases total network traffic proportionally to the number of deployment entities. This increase is due to the monitoring traffic that is proportional to number of deployer entities. Multicast injection total network traffic keeps constant, since it is not proportional to number of deployment entities, only to deployment requests rate.
Figure 6.14. Success Ratio vs. Number of Deployers

Figure 6.15. Success Ratio vs. Number of Deployers (High Load)
Figure 6.16. Total Network Bandwidth vs. Number of Deployers
6.5. Summary

Multicast injection deployment mechanism implementing resource-initiated allocation was compared to a service-initiated allocation deployment mechanism similar to Xweb prototype implementation. Simulations were carried out in which application networks were deployed in a number of resource nodes during a period of time with varying deployment request rates. Deployment success ratio, resource contention, unused allocations and communication costs were measured for a number of resource loads and deployer entities quantity.

Simulations of application network deployment with service-initiated allocation show that: deployers are complex and therefore have high computational and storage requirements. Simulations of multicast injection with resource-initiated allocation show it requires no discovery and monitoring traffic to deploy application networks; deployers are simple without resource monitoring, availability table, mapping calculation or resource configuration; service providers need to make public service specifications and resource requirements; and resource providers implement their own policies to control its usage; and need to interpret service specifications.

Analysis of simulation measurements shows that at low resource loads:

a) Multicast injection and centralized allocation deployment mechanisms behave similarly.

But at high resource loads:

a) Resource-iniciated allocation multicast injection has higher success ratio,

b) Resource contention for service-iniciated allocation deployment is notable, and increases with resource load,

c) Unused allocation for resource-iniciated allocation are not significant,

d) Bandwidth requirements at deployers locations are much higher for centralized allocation than for multicast injection, being proportional to resource nodes quantity,
e) Total Bandwidth requirements are higher for multicast injection, and increase with deployment requests rate,

Increasing monitoring rate of centralized allocation deployment mechanisms:

a) Increases success rate, but not more than multicast injection; it is due to resource contention decrease, but at the cost of higher total and per deployer bandwidth.

As the number of deployment manager increases:

a) Resource contention increases, and therefore success ratio decreases,

b) Also total network traffic increases.