

# Scalability analysis of Tycoon

## *Final Report*

A. Sottoriva  
openlab summer student  
CERN  
13th August 2007

### Abstract

*Tycoon is a market-based resource allocator for computing on-demand. In such system the users, with a minimal interaction, can buy and instantiate resources, while competing with the other customers. The general architecture of Tycoon has therefore to be scalable: here we will discuss the scalability issues that such architecture involves and we will study the possible bottlenecks of the system. The idea is to perform benchmarks on a set of machines in order to understand if the software is enough scalable to be reliable for several users at the same time.*

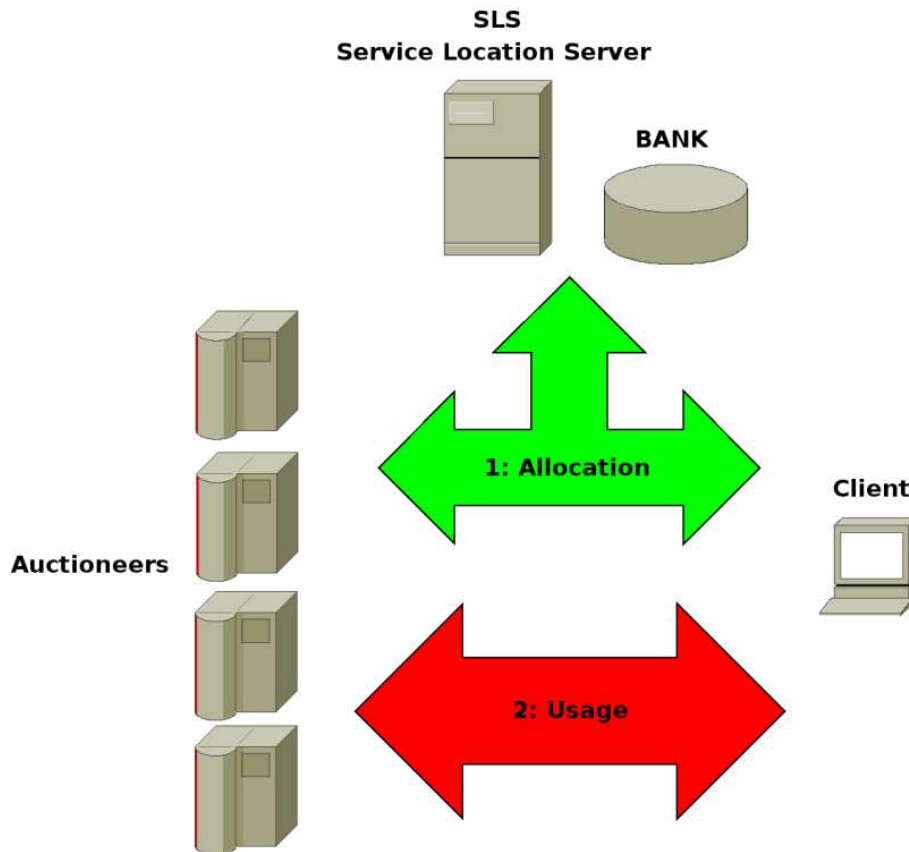
## 1 Tycoon's basic concepts

Tycoon is a distributed virtual machine manager, able to allocate remote virtual resources via a market-based system. The basic idea of Tycoon is to provide to the user a way to deploy and manage multiple remote virtual machines by betting for them a certain amount of credit. The scenario is however slightly different from classical auctions, when the user buy a resource in fact, it will be in constant competition with other hypothetical users that claim the same resource, having higher or lower amount of money. After a user has then got a resource (a percentage of the computational power of an auctioneer), Tycoon leaves the user to manage it by connecting to the allocated virtual machine. The virtualization software used by Tycoon to create VMs is *Xen* (<http://www.cl.cam.ac.uk/research/srg/netos/xen/>), Tycoon in fact deploys *Xen* virtual machines on which the user has administrator privileges. Once the resource have been instantiated, it start to be consumed while used, *eating* the money that was bet for it. Nevertheless, in this document we will not discuss all the market-based algorithms behind Tycoon. We will instead focus on the scalability issues that such system involves.

## 2 Architecture overview

The architecture of Tycoon, shown in figure 1, presents four main components: the client, the auctioneers (resource servers), the SLS (Service Location Server) and the Bank system.

We can distinguish two different steps on the creation of the resource: the *allocation* step and the *usage* step. The latter is suppose to be highly scalable just because there is no communication between different auctioneers (unless the user voluntary decided it), we will instead investigate the first step where are present the subjects discussed further.



**Figure 1.** Tycoon architecture overview.

## 2.1 The client

The client side needs a Tycoon client to perform basic operation such as resource allocation, resource deletion, bank account managing and resource information retrieving. The security plays also an important role, all the communication between the subjects is indeed encrypted and authenticated using RSA and DSA keys.

## 2.2 The auctioneers

Each of the physical machine that wants to provide resources has to run a Tycoon server. As we'll see later, each auctioneer has to be registered in the SLS (Service Location Server) to be known by the clients. Once the client has deployed the resources, it will use its authentication credential to access the virtual machines in the auctioneer individually, via *ssh/scp*-like protocols. The auctioneers therefore, are not responsible of any kind of communication between each other, or any kind of distributed synchronization process. It's in fact due to the single VMs to eventually build communication fabric, perhaps for a GRID system.

## 2.3 The SLS - Service Location Server

The *Service Location Server* represents the central resource database of the system. All the information about available physical machines, their technical characteristics and their costs are browsable using the SLS. The Tycoon client provides a simple and direct way how to retrieve information from the SLS server in order the user to

take decisions about betting. SLS doesn't directly deal with resource allocation, it just gives indication of which resources are available to the user and at which cost. Such a user will eventually then ask to the Tycoon system to allocate a VM on a specific physical host discovered using the SLS.

## 2.4 The bank system

A crucial part of Tycoon is the *bank*. The bank manages the virtual market that works behind the resource auctions: credit transfers, bids, funds, refunds, etc., however the technical details of such system are not actually known. The bank system is indeed the only part of Tycoon which is closed source. Moreover, and this is an important issue for scalability, the system is also strongly centralized. As we'll see is here that we'll focus our effort on benchmarking Tycoon's scalability.

## 3 The scalability issue

As we have seen, Tycoon looks like a pretty scalable architecture from the point of view of the resources. Whereas, it is also easy to guess the possible bottlenecks: the Service Location Service and the Bank. While the SLS presents no problems on adding redundancy and/or scalability, the bank system, centralized and closed by definition, creates an evident bottleneck for the entire architecture. Our chief objectives are therefore to stress this part of Tycoon with the largest request set our test bed can afford.

## 4 Scalability test structure

The way we will proceed with our scalability test is split in two main kinds of benchmarks: the resource allocation simulation (*tybench*) and the bank *get\_balance* stress test (*balbench*).

### 4.1 tybench

The structure of this first testing program is the following:

- Allocate the virtual machine on the selected host (stressful for the bank)

First a user selects a host among the ones available from the Service Location Server. He then bids for a virtual machine in such host. The bank first processes the bid and eventually, if the user is entitled to allocate the resource, the banks also contacts the Tycoon server on the specified host asking to deploy a Xen virtual machine. After the boot, the user has granted the access to the machine with root privileges.

- Copy the input data to the account (1 MB)

An hypothetical user is supposed to upload to the VM a certain amount of input files. Unfortunately the input data file in our simulation is not a realistic example: we in fact discovered a problem with the Xen kernel patch provided by Tycoon 0.5.0p366 which slows down dramatically the bandwidth (problem fixed on newer releases of the kernel patch for Xen). This makes not feasible to move a large amount of data to the destination host without spoiling the time scale of the benchmark. For this reason, together with the fact that we're concentrating on stressing the bank system, we're considering only 1MB large input file.

- Copy a CPU-eater script to the account

A user copies then the source code or the binaries of the software he wants to run on the remote account. In this case the software is just a CPU-eating script, discussed further.

- Execute some computation and generate output data

After the computation has taken place, an output file is generated, of the same size of the input file.

- Copy back the output

The generated file is copied back to the source host. Here we've the same problems of the input file transferring, the bandwidth problems make the simulation pretty unrealistic.

- Shutdown the virtual machine

Afterwards, the virtual machine must be shut down.

- Delete the deployed resource (stressful for the bank)

The account is entirely deleted. This process undeploys the VM and destroys all the data previously allocated. The bank takes then care of an eventual refunding and all other related money transfers.

## 4.2 baltbench

This second test is much simpler than the first one. It just consists on overflowing the bank of `get_balance` requests. The `get_balance` request gives as output to the user the actual amount of credit in his/her account. The purpose of using this command is that the amount of information is quite small, therefore our test is not spoiled by bandwidth problems such the ones we are going to see for Xen-patched kernels.

## 5 The testbed at HP

HP-Labs in Palo Alto provided us a testbed of 56 machines, equipped with Fedora Core 4, Xen 3.0.2 and Tycoon 0.5.0p366. Unfortunately, due to several technical problems, test performed on the entire set of computers had failed in some of them. Hence, cause to the unreliability of some of the machines, the effective testbed was reduced to 32 servers with different characteristics. The information available about the architecture of the test site are not detailed, we however consider the following statements as true:

- the machines have all the same bandwidth
- they don't share filesystems that are used, directly or indirectly by Tycoon (e.g.: no VM images sharing)
- no other relevant job is performed by those machines during the tests
- from the point of view of the bank performances, the machines have the same behavior

Despite that having a remote testbed on HP may sound convenient, this configuration has many limitations. The main is that we do not have any privileged access to those machines, we therefore don't have any chance to tune the software to adjust it to our purposes. Concentrating the request only in the bank system in fact, by for example modifying the Tycoon server, avoiding the deployment of the images, could lead to a more accurate test, not spoiled by the effective deployment of virtual machines. Moreover, we are limited with the number of virtual machines per physical host. Each further VM we allocate per host increases significantly the account creation time,

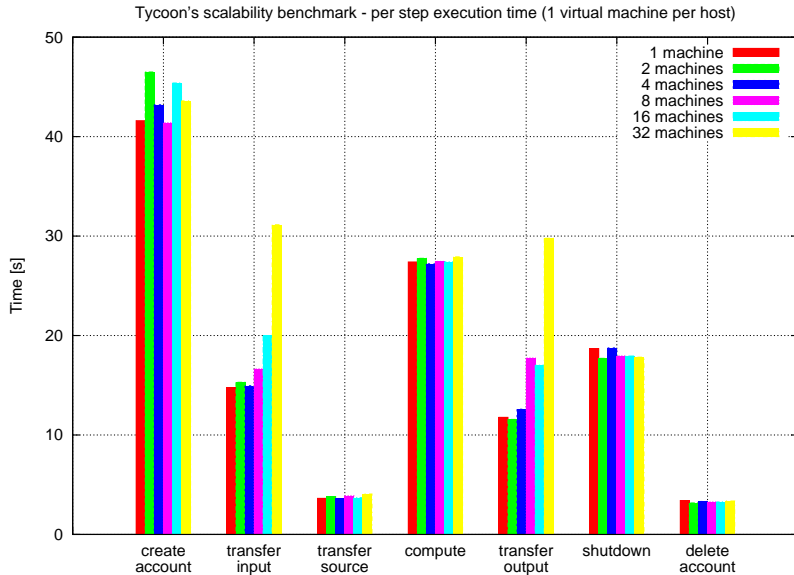
and the dynamics behind the resource allocation could easily spoil the test. At this point is reasonable to argue also about the size of the testbed. Tycoon is indeed expected to be used for large scale systems such as hundreds or perhaps thousands of users and here we're testing the application up to 64 simultaneous clients. Nevertheless, the aim of the test was starting to explore the problem, investigating the scalability also for a limited number of requests. Once the scalability at this level of magnitude will be ensured and the system will be considered reliable, it will be relatively straight forward to extend the test to a larger set of machines.

## 6 Results

The 1VM-per-host *tybench* test was run from a single machine with a single Tycoon bank account while the 2VM-per-host test was executed using two different Tycoon users from two different machines of the CERN computing center. For the *histbench* test the situation is analogue to the first *tybench* test.

### 6.1 1 VM per host

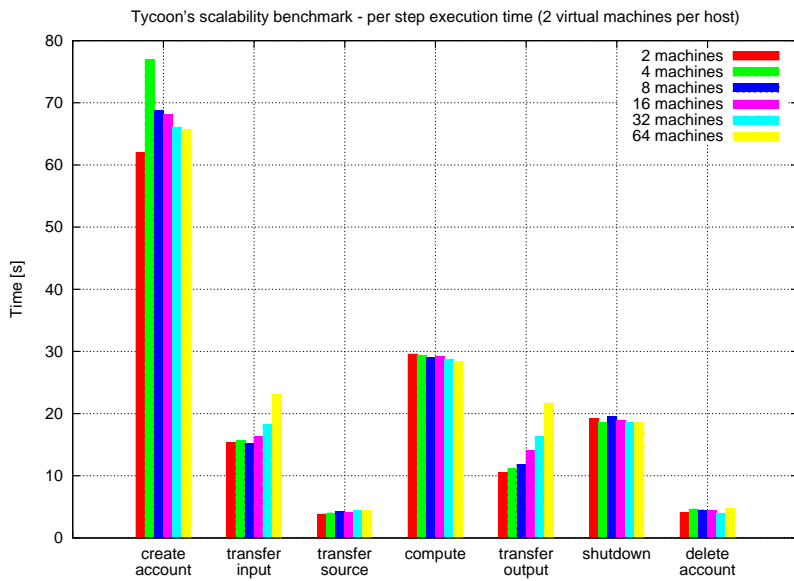
The graph in figure 2 shows, for every single step of the benchmark, how Tycoon performs for 1,2,4,8,16 and 32 machines. Here we've 1 virtual machine per host, so we have no resources contention problems for a single machine. Despite that the account creation has a significant impact in the overall performances, we can observe that even up to 32 machines the software scales properly, as well as for the account deletion step (the two main processes that are stressful for the bank system). Is easy also to notice the behavior of the network transfers steps: input and output files transfers. The process scales exponentially, however it's not related to Tycoon cause we've to bear in mind that we're using a single machine to delivery all the requests. The bandwidth is therefore limited to a single host bandwidth.



**Figure 2.** Inter-steps performance of *tybench* with 1 virtual machine per host.

## 6.2 2 VM per host

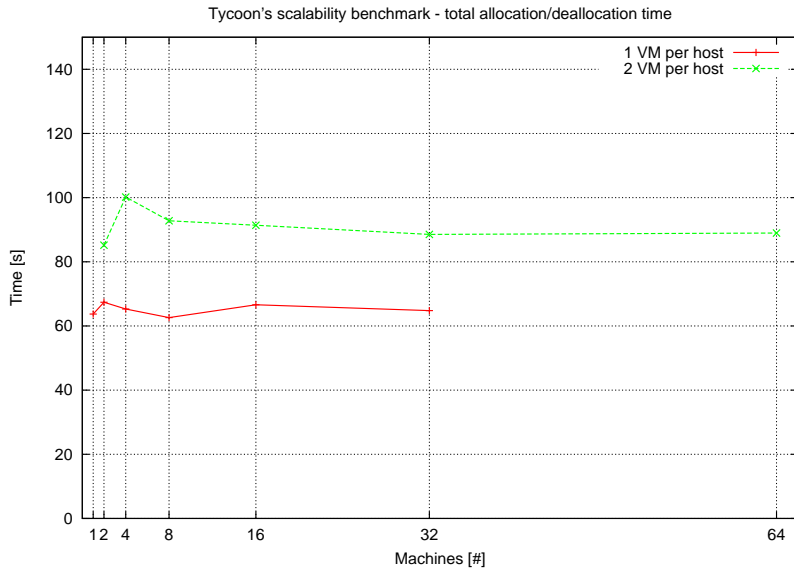
The situation for 2 virtual machines per host is similar. We can recognize the same pattern for the input/output transfers and also the same scalability characteristic of the first test. A significant difference relies in the account creation step: for obvious reasons the deployment of a single image, with respect of multiple images, needs more resources. As shown in figure 3, the VM allocation cost (*create account*) is now increased of roughly 45%, this implies some scalability issues from the point of view of the VM density per host. Despite this, the latter aspect is not investigated in this paper cause is not among the objective of the study. Regarding the other steps instead we can observe a positive result: all of them had remained approximately constant with respect to the previous plot with 1 VM per host.



**Figure 3.** Inter-steps performance of *tybench* with 2 virtual machines per host.

### 6.3 Total allocation/deallocation performance

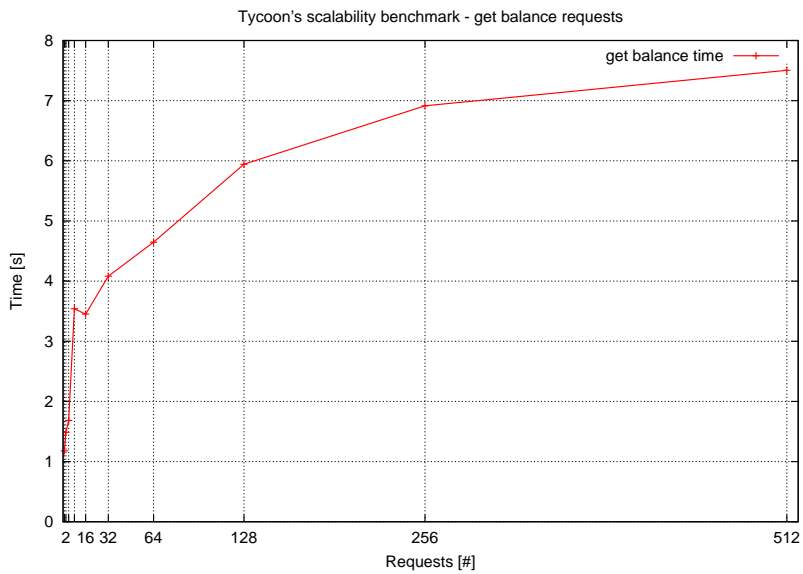
As global view, the plot in figure 4 shows explicitly what it was clear in the previous plots. Here the times of the crucial steps involving the bank and in general the deployment of the resource are added (*account creation*, *account shutdown* and *account deletion*). Those information give a clear picture of the scalability of the entire system. As we can see in figure 4 then, the execution time can be considered finally constant up to 64 account creation requests at the same time.



**Figure 4.** Total allocation/deallocation performance of *tybench* with 1 and 2 virtual machines per host.

### 6.4 Bank balance request stressing

The plot in figure 5 shows a clear logarithmic trend, however we remain still in the order of tens of seconds even with an exponential increment of the number of requests. The curve becomes flat quite soon and it can therefore be considered a very good result. Moreover this non perfectly flat behavior could probably be caused by the bandwidth contention or the rising number of Tycoon clients in the source machine. These results are then coherent with the previous ones and confirm the scalability of the Tycoon architecture, at least up to the order of magnitude of about 100 of users.



**Figure 5.** get\_balance requests scalability.

## 7 Conclusion

We studied the Tycoon architecture from the user's point of view and we found its critical bottlenecks. We then investigated the scalability of the entire structure up to a hundred of users, using the HP-lab testbed in Palo Alto. We then went into some virtualization issues in order to understand some underlying processes that take place behind Tycoon. Along the way we also found an important bug (now fixed) of the bank system, consisting on a wrong calculation of the refunds after the resource deletion. Such bug was producing credits on the bank accounts of the users which was growing after the resource consumption. Concluding, we developed benchmark scripts to perform our investigation, finding their advantages and their limitations. We had positive results regarding scalability in all the tests executed. We finally can state that up to the scale we took into the account (about a hundred of possible simultaneous users) the Tycoon system scales perfectly, even presenting possible bottleneck in its structure.

## 8 References

- [1] *Tycoon homepage*, <http://tycoon-dev.hpl.hp.com/>
- [2] Kevin Lai, *Tycoon User's Manual*, [http://tycoon-wiki.hpl.hp.com/tycoon/doc/users\\_manual\\_en/](http://tycoon-wiki.hpl.hp.com/tycoon/doc/users_manual_en/). 2007.
- [3] Kevin Lai, *Tycoon Administrator's Manual*, [http://www.hpl.hp.com/research/tycoon/documentation/admin\\_manual\\_en/index.html](http://www.hpl.hp.com/research/tycoon/documentation/admin_manual_en/index.html). 2007.
- [4] *Xen homepage*, <http://www.cl.cam.ac.uk/research/srg/netos/xen/>
- [5] The Xen Team, *Xen Interface Manual*. Cambridge 2005.