Article 4 of the series *How to apply performance modeling*

The dynamism of software resources

This is the fourth article in the series which sheds light on what is possible with performance modeling and what it takes to apply it. The paramount purpose of performance modeling is to explain and predict application performance.

So far we have seen what inputs a performance model needs and how some basic analysis scenarios can be conducted. In the previous article among other things we had a look at hardware resources. In this article the focus is on software resources.

What is a software resource? It is not easy to give a general definition, but here are some examples:

- A set of processing threads on a multitasking server
- A set of HTML sessions
- A pool of persistent network connections
- A database lock
- A single threaded software module
- A critical section in an operating system

The performance behavior of software resources is very interesting they make application performance more dynamic, but this is not easy to explain let alone to predict. It is no surprise that sometimes reference is made to mysterious “hidden bottlenecks” that turn out to be ill dimensioned software resources. Logging with usable metrics is seldom available. Try to get answers to questions like: What is the utilization on a single threaded software module, or a connection pool. Or what is the delay they add to the response times? Forget even about predicting there impact . . . unless you use a performance model.

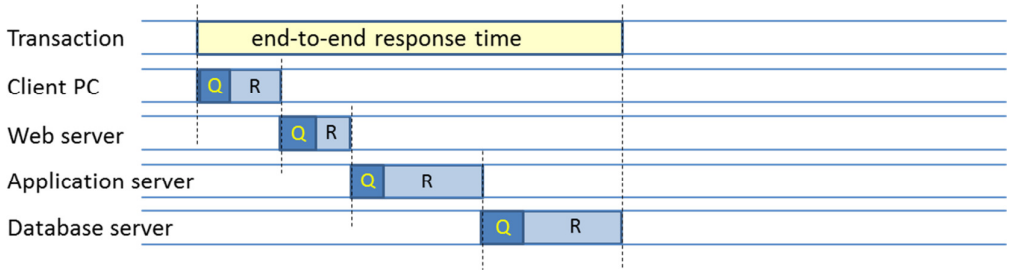
It is difficult to explain the performance behavior of software resources without going into the details of models. So since the aim of the articles is not to treat the models but how to apply them and since I promised not to use math in these articles, here is an impression first, followed by an example of an analysis scenario with the model.

By the way I earlier published an example of analyzing an application with single threaded software modules: “Case Study Securing System Performance with Transaction Aware Performance Modeling”. You can download it from either the MeasureIT or our website: www.mbrace.it.

An impression of software resource performance behavior

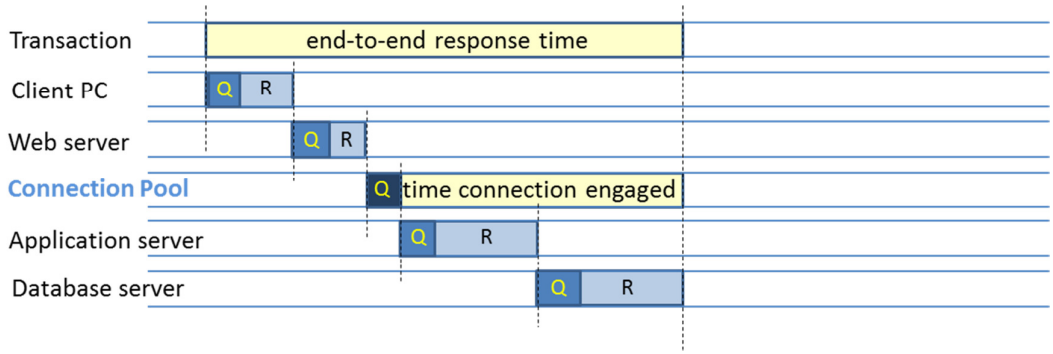
The example concerns a network connection pool with 10 persistent connections between an application server and a database server. When the application needs to access the database it fires off a call and to this end it first takes a connection from the pool. Then the connection is used to exchange data between the application server and database server. When the call is completed the connection is given back to the pool again.

The next figure shows what the average response time looks like when there would be plenty connections.

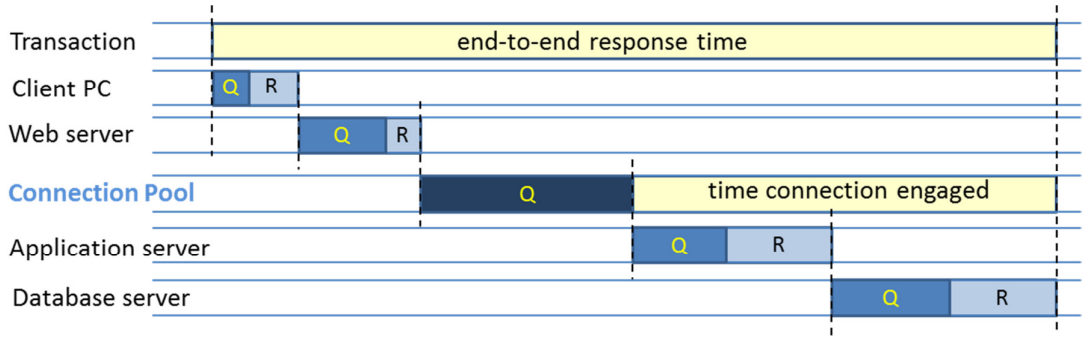


The response time is built up by the times spent on the PC and the servers. To keep the example as simple as possible, no networks are taken into account here. The times on the PC and servers are composed of time spent on the hardware resources (R) and time spent by waiting for them (Q). The resource and waiting times can be split between CPU and disks. The waiting time depends non-linearly on the transaction volume and the time the resource is engaged (R). When the transaction volume of the application grows, the waiting times (Q) may increase, but the resource times (R) remain unchanged. These hardware resources are load-independent resources.

The next figure shows how the response time is built up when the connection pool involved has only 10 connections. When multiple users are served concurrently also multiple connections are engaged up to the maximum of 10. When all connections are engaged and a call to the database must be made for a new user the application has to wait until a connection is released again. The connection pool as any software resource has a utilization percentage too, just like the hardware resources. E.g. when 8 out of the 10 connections are engaged we have an 80% utilization.



Since the waiting time depends non-linearly on the time the connection is engaged it is interesting to see how this is determined. The above figure illustrates that this is determined by the times spent on both the application server and the database server. This time is taken already into account for the response time via the hardware resources of the application and database server and does not change the length of the response time, however the waiting time does. Consider the hardware resources of the application server and the database server and the underpinning hardware resources of the software resource, the connection pool. The next figure shows what happens when the load increases.



When the usage volume of the application increases, consequently the queuing for the hardware resources increases. So the time spent on the underpinning hardware of the software resource increases. In other words the time the connections are engaged increases. Without that the queuing for the connections would already increase by itself. The fact that the engagement time of the connection also increases, causes the queuing for the connection pool to further increase even more. Notice that when the database takes considerable more time incidentally, the engagement time of the connections increases as well and may cause further increase of the queuing. So the software resource introduces extra sensitivity for the performance of the application. Since the engagement time of the software resource varies with the load it is called a load-dependent resource.

Notice that no extra measurement data is required for the software resources in the model. Introducing it into the model can simply be done via the data of its underpinning hardware.

Analysis scenario

To illustrate how the impact of the software resource can be handled by a performance model, an analysis scenario is conducted on an application that runs on an infrastructure chain consisting of PCs, Wide Area Network, Web servers, Application servers and a Database server. In this example, for the sake of simplicity, only the Application server, Database server and their interconnecting LAN are represented in the model. For the same reason the application has only one transaction type. The limits for the Apdex Index are set at 2 and 4 seconds. Both servers have 4 CPU's and each has only one disk. There are LANs with 0.1 Gbps between the servers. A connection pool with 10 connections spans the Application server, LAN and Database server.

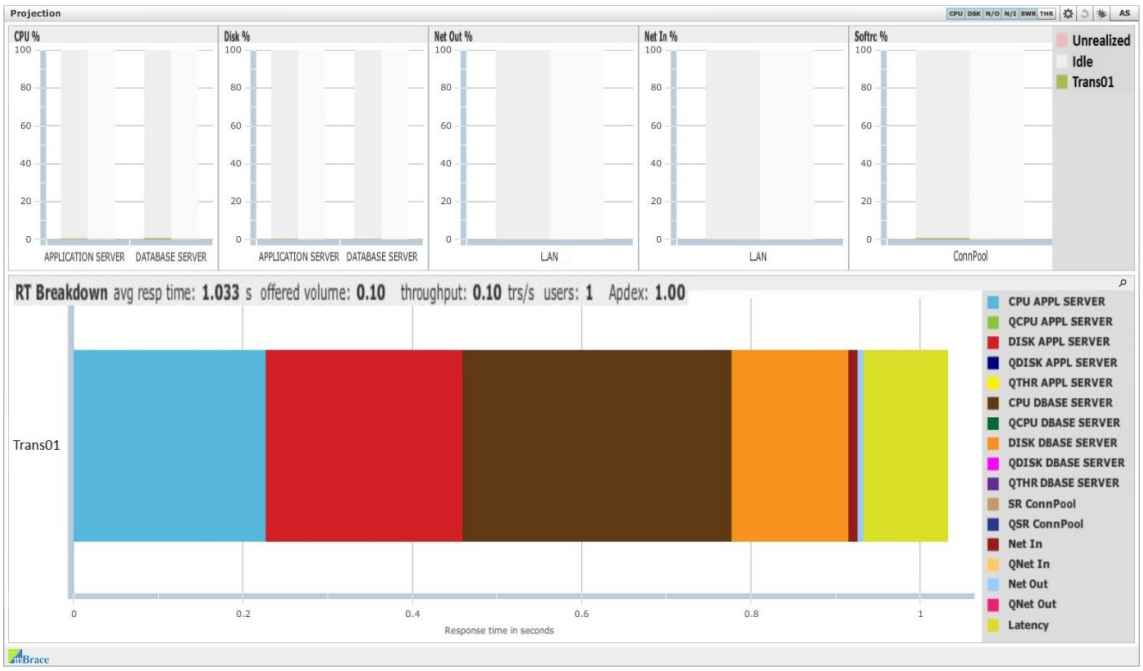
The Apdex index

This is what Wikipedia tells about Apdex:
"Apdex (Application Performance Index) is an open standard developed by an alliance of companies, the Apdex Alliance. It defines a standard method for reporting and comparing the performance of software applications in computing. Its purpose is to convert measurements into insights about user satisfaction, by specifying a uniform way to analyze and report on the degree to which measured performance meets user expectations."

The Apdex Index reports a figure between 0 and 1 as a measure of user satisfaction with the response times. 0 is unacceptable, 1 is excellent. The figure is obtained after setting two limits. Each response time below the lower limit is considered satisfactorily by the user. Each response time above the upper limit is unacceptable for the user and response times in between are tolerated.

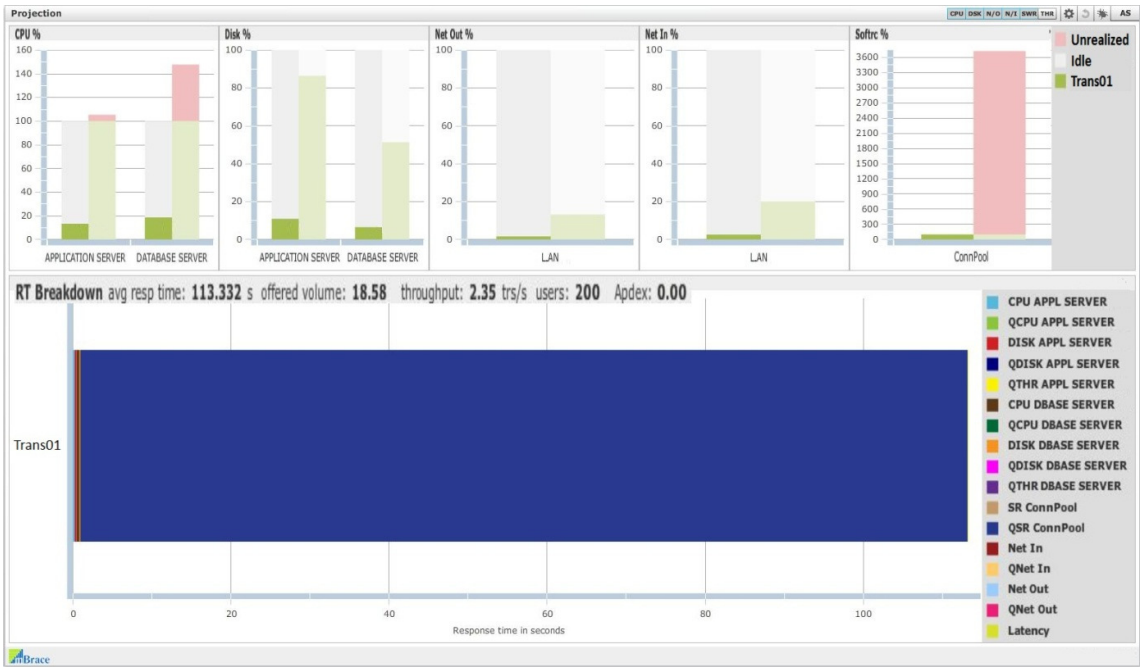
The Apdex Index is determined by: (#satisfying +(#tolerable/2))/#all responses.

Step 1. Single user performance.



With a single user we can see the transaction profile in the performance DNA. The minimum response time is 1.033 sec. All resource utilizations are (almost) 0%. The spent on the various resources of this part of the multitier infrastructure chain is more or less equally divided except for the outbound and inbound parts of the LAN which take much less time. Notice that only part of the infrastructure is represented in this model, consequently only part of the end-to-end response time is shown. I.e. the part that is relevant for exploring the performance of the connection pool.

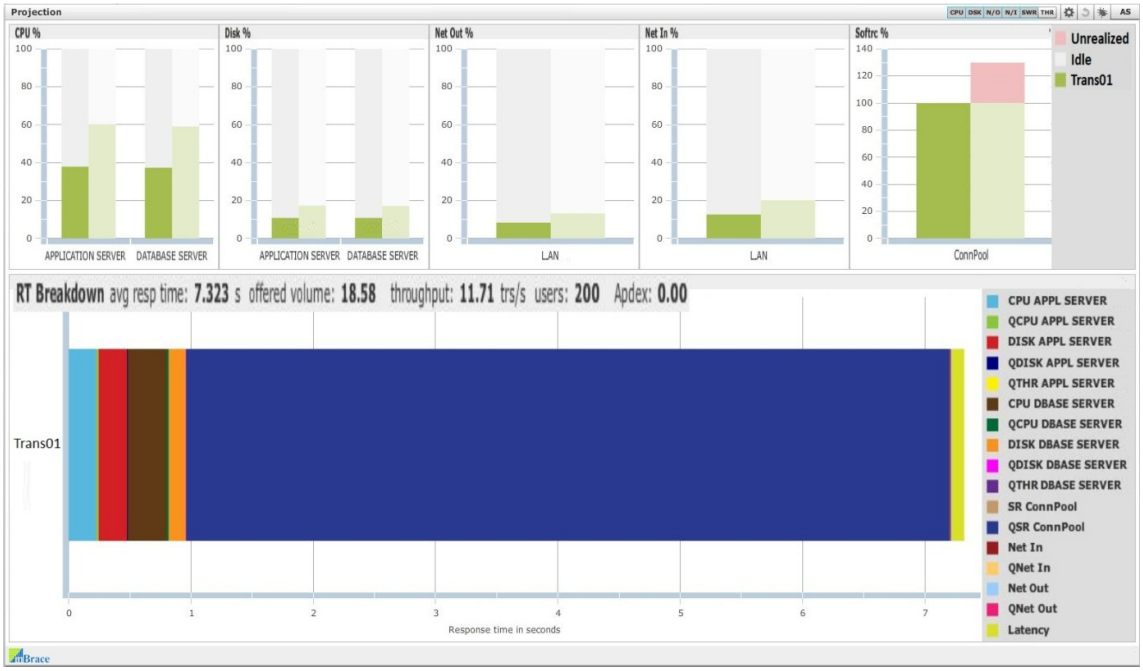
Step 2. Target usage volume



The above figure shows the application performance at the target load of 200 users. Obviously there is no way this configuration can handle the offered workload. The chart shows an average response time of 113 seconds, completely dominated by the blue color of the waiting time for the connection pool (**QSR ConnPool** in the legend). This is a color newly added to the Performance DNA. From the offered volume of 18.6 transactions per second only a meager 2.4 transactions are handled per second.

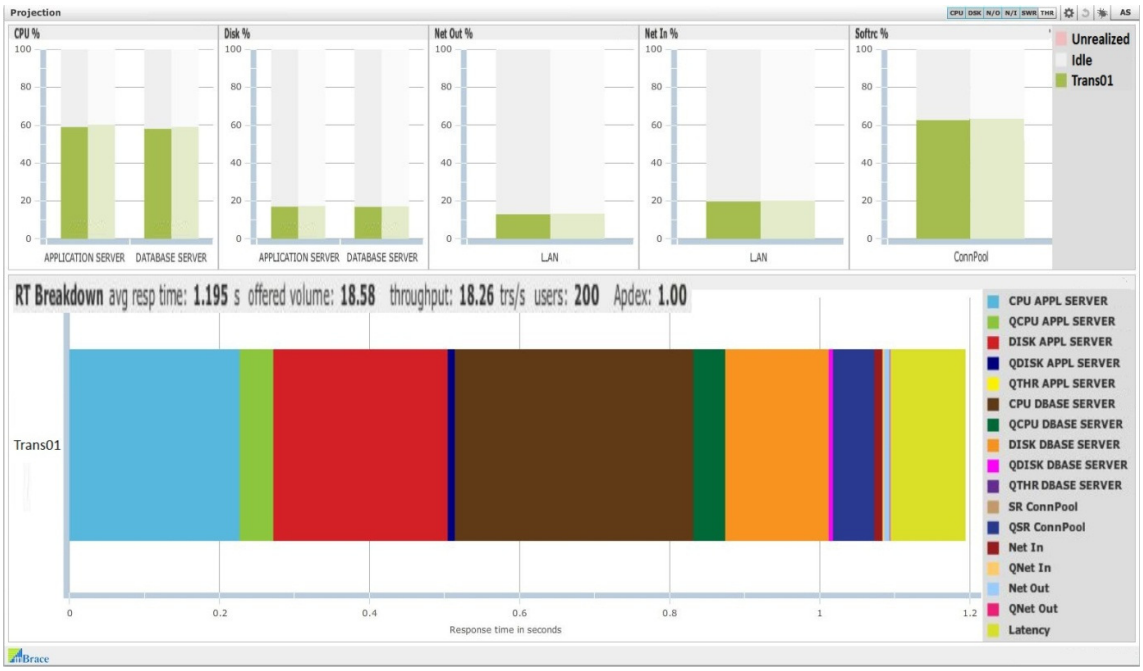
It is plain to see that there are three bottlenecks: the CPUs of the Application server, the CPUs of the Database server and the Connection pool. The Connection Pool is loaded at 100% seeking a utilization of over 3,700%. The CPUs are underutilized at 13% and 19% because of the other bottleneck at our software resource the Connection Pool, otherwise they would be overloaded seeking 105% and 148% respectively. Without the bottleneck at the Connection Pool they would be utilized at 100%. Obviously the Apdex index equals 0.0.

Step 3. Hardware upgrade



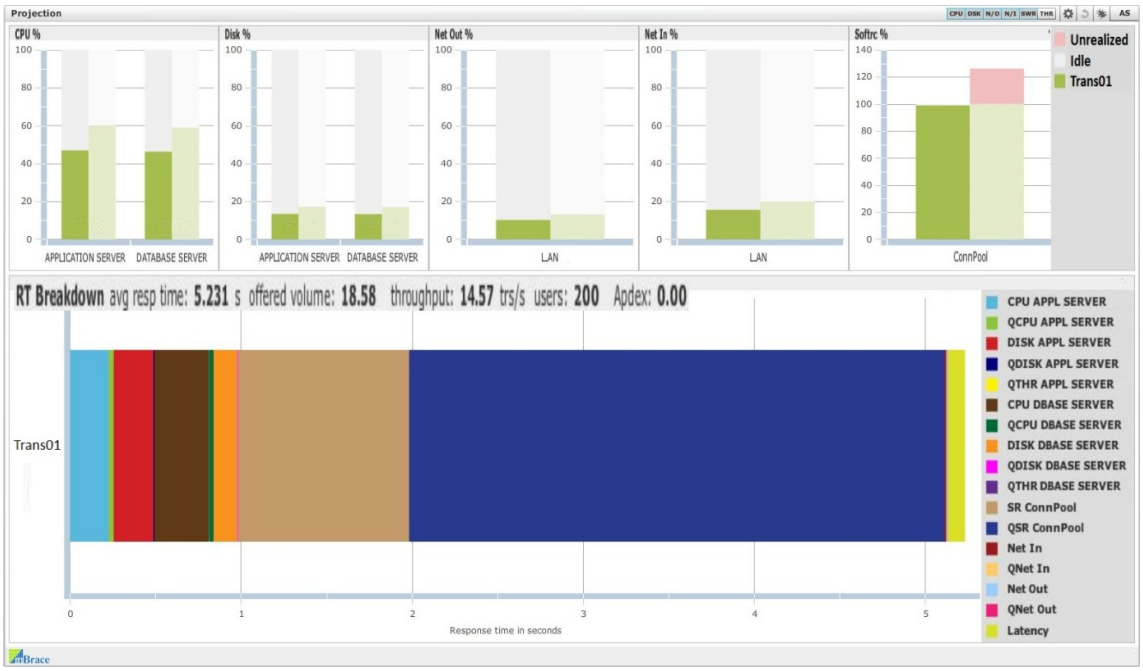
We already know that we have bottlenecks at the CPU's of the servers and at the Connection Pool as well. It is good practice to start with scaling the hardware. So the CPU's are horizontally scaled from 4 to 7 and 10 pieces respectively. The numbers of spindles are scaled from 1 to 5 and 3 respectively. The figure above shows that the response time improved a bit but the blue of **QSR ConnPool** is still dominant in the response time breakdown. The throughput climbed from 2.35 to 11.7 transactions per second; however performance is still not what it should be. The Apdex Index dropped to 0.0. The CPU's seek higher utilizations but are held back by the bottleneck that still remains at the Connection pool. Surprisingly though the Connection pool was not altered, its unrealized utilization decreased from 3,700 % to 130%. This is all and only due to the decrease of CPU and disk utilizations which causes a reduced engagement time of the connections (see section Impression).

Step 4. Enlarge connection pool 1



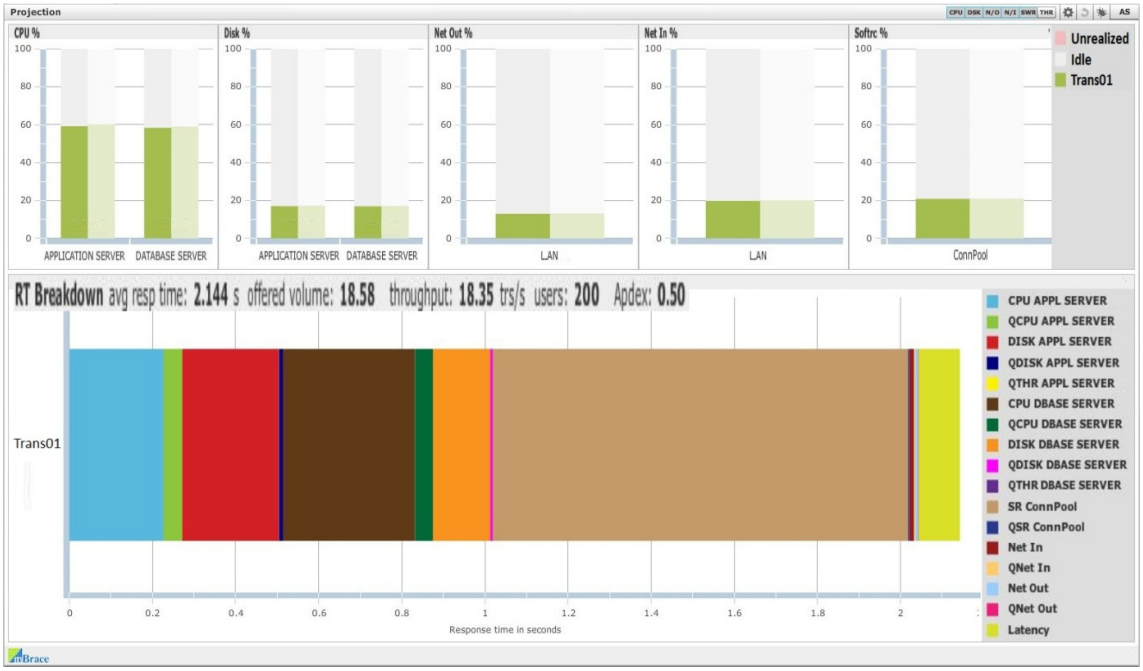
In this step in our analysis scenario the Connection pool, our software resource, is widened from 10 to 25 connections. As a result the performance now looks fine with a response time at 1.195 sec short enough to have the Apdex Index up at 1.0 again. Some colors newly added to the DNA modestly show themselves now. These colors can be found in the legend to the right with names that start with a Q. The **green** to the right of the **light blue** for instance reflects the time spent on queuing for the CPU's of the Application server (QCPU APPL SERVER in the legend). The throughput is only slightly less than the offered volume and resource utilizations are nicely on their targets.

Step 5. Introduce disturbance



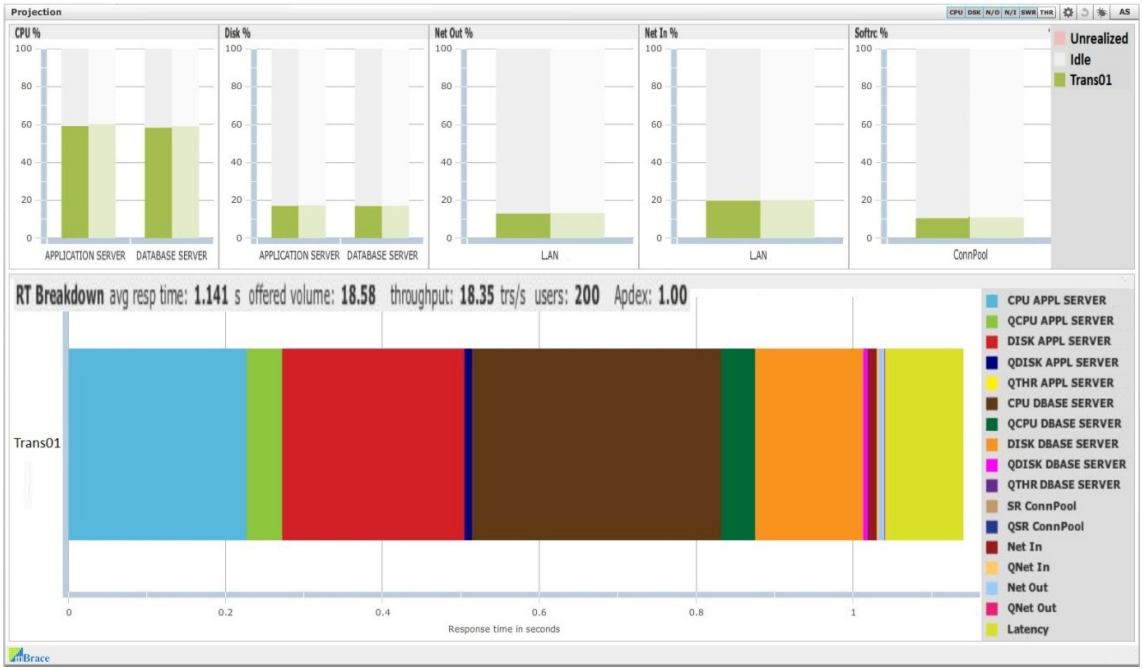
Software resources are sensitive for disturbances in the multitier infrastructure chain of the application. To test this, an extra delay of 1 second is introduced as such a disturbance at the underpinning hardware of software resource. The model shows that the performance is substandard again; Apdex is back at 0.0, for a response time of 5.2 seconds. Only 14.6 transactions are handled per sec from the offered volume of 18.6. CPU utilizations are kept low at 47% again by the bottleneck at the Connection pool. The response time increase is caused by the disturbance delay of 1 second (the **light-brown** part) and the more than 3 seconds waiting time for the **Connection pool** (blue). Obviously there are not yet enough connections in the pool to cope with a disturbance of 1 second.

Step 6. Further enlarge connection pool



The connection pool is further enlarged to 150 connections. The above figure shows adequate performance, though due to the response time increase of the disturbance, the Apdex Index only increased to 0.5. However the performance of our software resource is now stable for relatively large disturbances like 1 second.

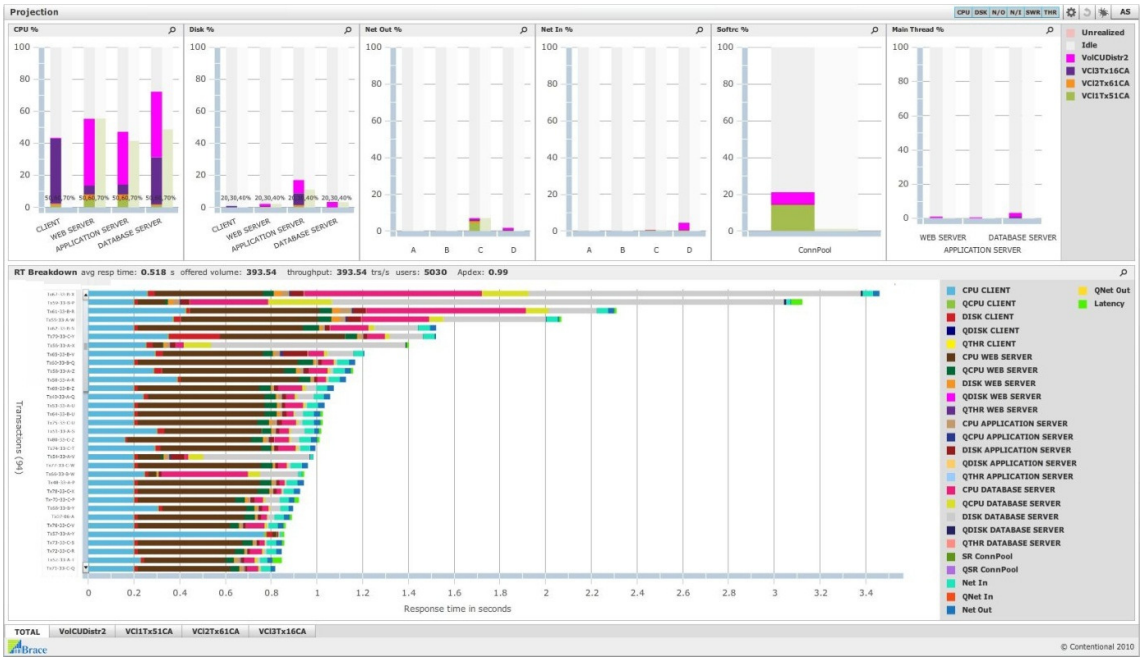
Step 7. Without disturbance



Finally the performance without disturbance is OK with a response time of 1.1 second, Apdex Index at 1.0 again and throughput is only a trifle lower than the offered volume. This also shows that it is good practice to maintain the utilization of software resources at a target as low as 10%.

Analysis of software resource impact with Transaction Aware Performance Modeling

Of course this not only is possible with single transaction applications. The same interesting analysis scenarios can be done transaction aware on applications with a multitude of transactions on an extensive multitier hardware chain as the next example shows.



Any questions, debate, criticism? Email: michael.kok@mbrace.it.