C-EMS Performance

Performance is generally a key factor in the evaluation and selection of technological solutions, and enterprise mashup platforms are no exception to the rule. Even after the solution has been selected, the question of server sizing arises again with each new project.

There is no simple answer to this question. To illustrate why, let's take the example of another question: "How fast does a car go?"

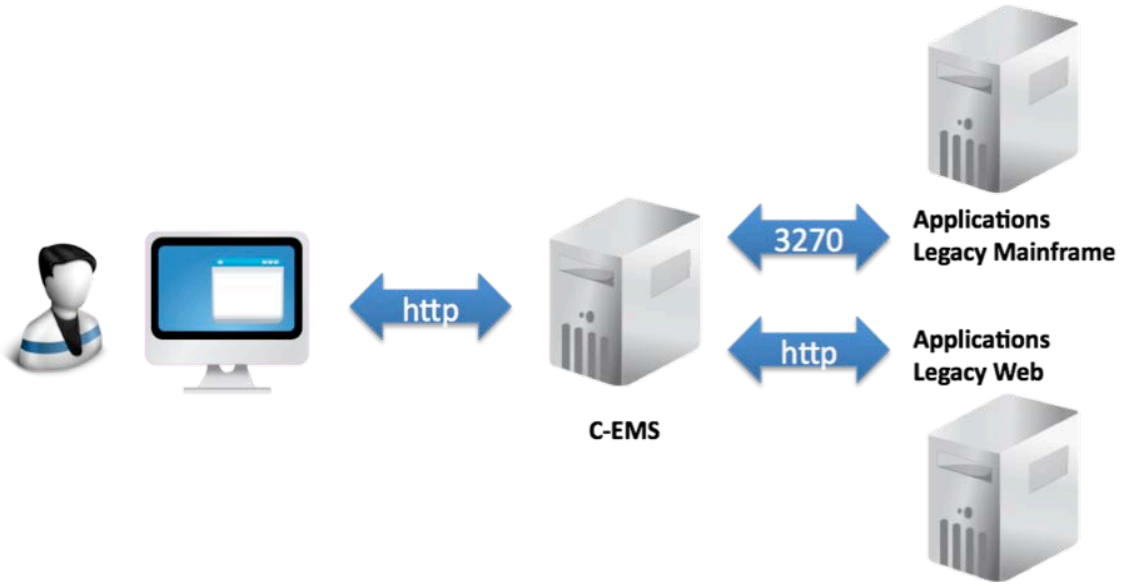
> ***The speed of a car depends mainly on its intrinsic characteristics, such as its engine power, total loaded weight and aerodynamic coefficient (Cx). If we wanted to be very precise, we could factor in other parameters: the way the paint quality affects the airflow capacity around the car, the nature of the exhaust system, the fit between the tires and the terrain… Numerous criteria external to the vehicle also play a role: the type of road surface, the weather conditions, the driver's experience, etc.***
>
> ***The question itself needs to be more specific: it all depends how we measure "fast". In terms of absolute top speed? Or the time from 0 to 60 mph? Or the time over one mile, from a standing start? Or the maximum speed reached in 10 seconds? These new questions touch on factors like gear range, torque and many other parameters.***

The performance of an enterprise mashup platform depends on at least as many parameters and can likewise be measured in different ways: the response time of a given sequence or transaction, the maximum number of transactions that can be run simultaneously on a given server, etc.

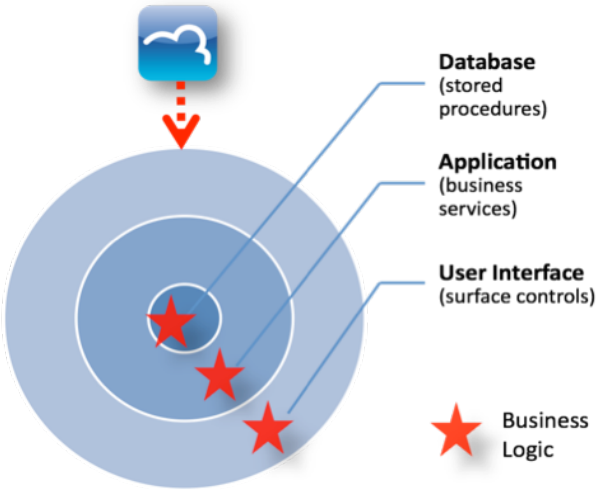
It is possible, however, to outline an answer, but this requires going into the details of how Convertigo executes transactions and understanding the various steps involved. The memory consumption and simultaneous transaction processing aspects must also be considered. Once these elements are established, it is possible to size the hardware correctly and to implement the software architecture best suited to a successful Convertigo deployment.

Performance Chain

Convertigo transactions and sequences are part of a chain of IT components that include, starting from the user: the browser, the terminal, the Internet access, the Convertigo server, the legacy applications, and the servers that host the applications.



Convertigo calls up the screens from the legacy application, in much the same way as a user might do so manually. This means that nothing needs to be redeveloped: you can access the applications as they are, and retrieve all the available business logic in the user interface (surface controls), the application (data validation, business services) and the databases (stored procedures).



C-EMS receives requests from users and activates the various legacy systems, which continue to run in the same way as before. It acts as a kind of proxy, aggregating, filtering, converting and publishing the initial data in a modern mashup.

C-EMS is, in a very real sense, an intermediate server between the user and the legacy applications. The response time perceived by the user therefore includes the response times of the legacy applications plus the Convertigo response time, as well as the time required to serialize and carry the information to the end browser.

This intermediate layer inevitably involves an additional “overhead” time, therefore a Convertigo transaction cannot, as a rule, be faster than the original transaction in the legacy application.

NB: If the Convertigo cache system can be enabled, the response times perceived by the user may be better than those of the original application, because Convertigo no longer calls the legacy applications once the data has been retrieved. For more on this topic, see the *Cache* section at the end of the document.

Web Transactions

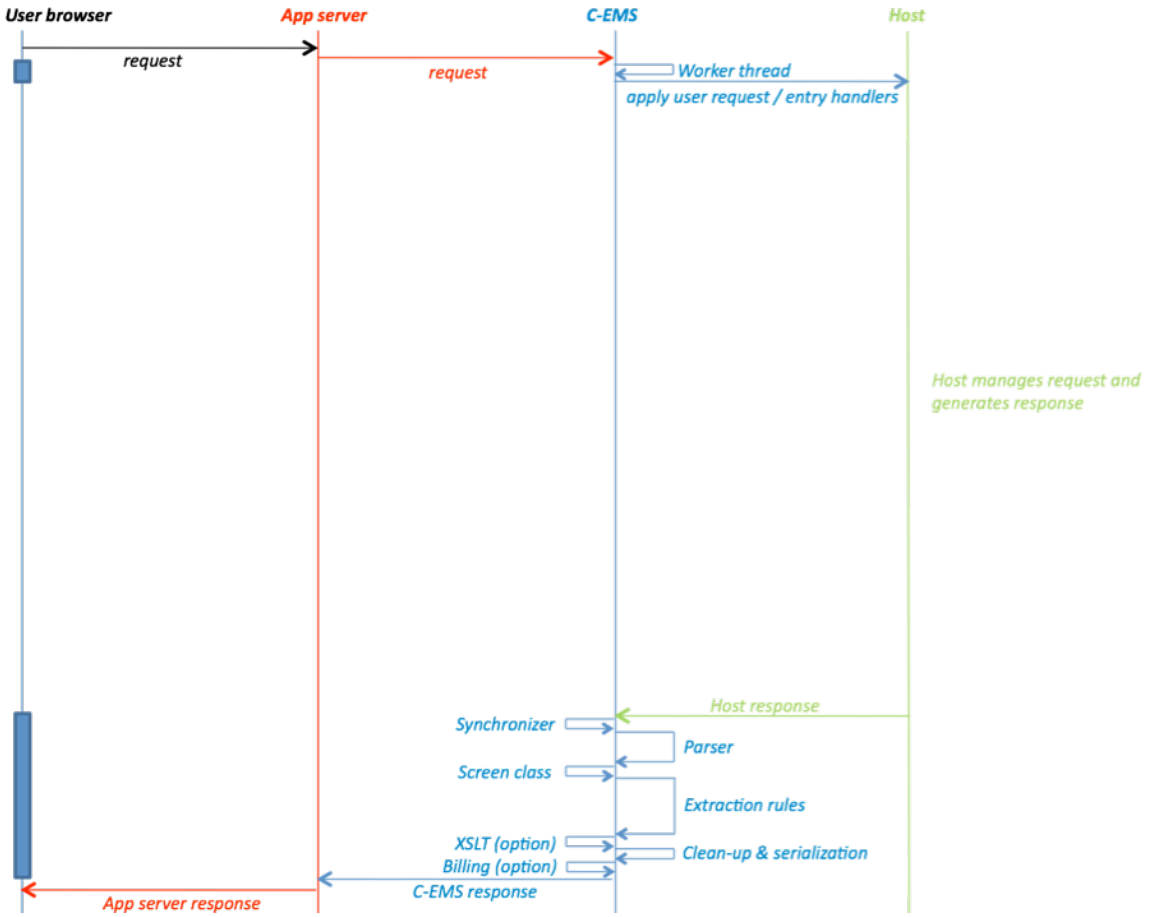
NB: All of the response times mentioned below are average values observed in test or production environments, on "standard" Intel machines, i.e. a 2GHz dual core processor with 2 to 4 GB of RAM.

NB2: All of the response times used below for illustrative purposes are based on web pages of average complexity, such as for example a page of Google search results.

When Convertigo is used to access an existing Web application, the following steps are implemented:

- The request is processed on the Convertigo server (Tomcat, for example):
  - This time is typically very short – less than 5 ms.
- The request is analyzed by Convertigo:
  - Ditto.
- Start Worker Thread:
  - Convertigo manages its own pool of threads. The default limit is 100 threads per instance.
  - This time is typically very short (< 5 ms), except when the server starts to get overloaded (see the *Simultaneous Transactions* section).
- Apply User Request:
  - Convertigo applies the actions required by the user request, typically filling in input fields and clicking on buttons.
  - The response time may amount to several seconds, depending mainly on the complexity of the target page.
  - At this stage, the Convertigo overhead is no more than a few milliseconds.
- Wait for response:
  - Convertigo needs to detect that the response has been received in full (a crucial synchronization mechanism).
  - Once again, most of the response time is down to the initial application – the Convertigo overhead is very small.
- Parse response:
  - The incoming HTML is converted into DOM format.

    - Convertigo has 2 algorithms for this purpose:
        - Fast Parser: relies on result buffering in XUL, followed by Java parsing. The response times depend on the complexity of the response page received, with average times ranging from 15-20 ms on a powerful computer to 50-100 ms on other machines.
        - Classic Parser: the previous method doesn't always work, due to bugs in the XUL buffering. In these cases (approx. 5%), a Java DOM has to be generated from the XUL DOM (in C). This technique is somewhat slower.
- Detect screen class:
    - This time is typically very short, less than 5 ms.
- Execute extraction rules:
    - This time depends on the complexity of the rules and the volume of data to be processed.
    - A basic rule takes between 20 to 80 ms.
    - A complex table rule can take up to 1000 ms.
- Apply a server-side XSLT transformation (optional):
    - This step is usually executed client-side, but if it needs to be done server-side, it can be expected to take about 40 ms on average.
- Serialize output DOM and clean up resources
    - This time is typically very short, less than 5 ms.
- Execute billing classes (transaction count, duration, etc.) (optional)
    - This time depends mainly on the technique adopted by the developers to store billing information.



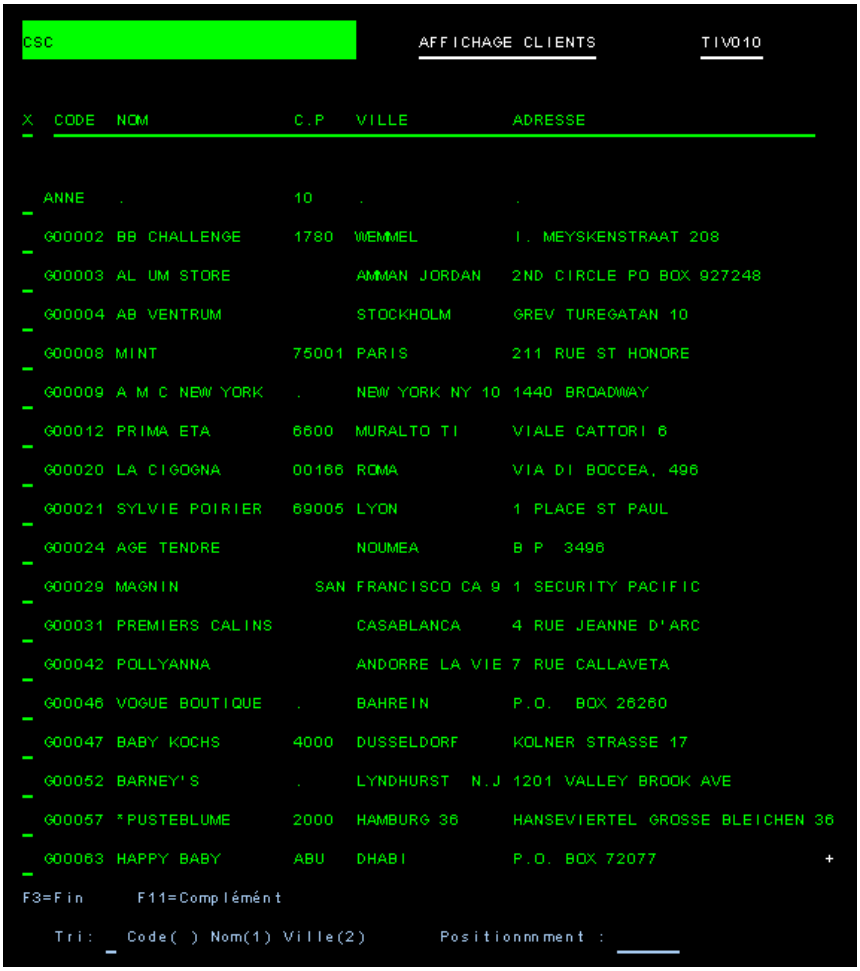
Web Access Overview

In conclusion, cumulative Convertigo times rarely exceed 200 ms in general for one transaction, whereas the response times of the target applications are often much longer (up to several thousand milliseconds).

The Convertigo overhead is therefore generally insignificant compared with the normal response times of the applications. This remains true even under unfavorable conditions where the C-EMS response times become longer, as this is principally due to responses from target applications which themselves are more complex and/or more voluminous and therefore more time-intensive.

Legacy Transactions

NB: All of the response times used below for illustrative purposes are based on legacy screens of average complexity, such as the following AS400 screen (a classic customer management system, with a data table and function-key actions):



Where Convertigo is used to access legacy applications (mainframe, AS400, Unix), the steps are basically identical. In this case, clearly, screen parsing is not based on XUL, but on analysis of the screen formats concerned (3270, 5250 and VT standards). As these formats are less complex than HTML, the parsing times are generally shorter in legacy mode than in Web mode, and are usually less than 10 ms.

For the same reasons, the response times for extraction rules are also significantly shorter.

Synchronization

The only step that is more complex in legacy transactions is end-of-page detection (synchronization). While the legacy screen frames always contain an end marker, this only really serves to indicate the unlocking of the keyboard, and there is no guarantee that the screen consists of just one frame. In 95% of cases, screens are limited to a single frame (CICS screens, for example), but there are situations in which a screen is made up of several frames. A detection loop is therefore required to pick up *keyboard_unlock* signals, to ensure that we really have reached the end of the screen.

Legacy Access Overview

Overall, the processing of a legacy screen is estimated to take an average of about 50 ms (compared to 200 ms for a web page).

Simultaneous Transactions

Multi-Thread Architecture

As we have seen, a Convertigo Web transaction can take up to 200 ms on average (or 50 ms for a legacy transaction), on a standard Intel configuration, by a fairly conservative assessment. This result suggests that approximately 5 simultaneous Web transactions (or 20 legacy transactions) can be handled per second and per processor core, and therefore about 8 simultaneous Web transactions (or 34 legacy transactions) on a dual-core processor.

As the computer CPU reaches saturation point (near to 100%), the execution time for the *"Start Worker Thread"* step (see the *Web Transaction* and *Legacy Transaction* sections above) rapidly becomes longer and longer. This phenomenon needs to be taken into account when sizing the servers – one should avoid approaching CPU saturation.

XUL

Convertigo uses the Gecko / XUL parsing engine to analyze Web pages. This engine is developed by Mozilla and supports the analysis of most existing web pages.

It can read very complex pages, but it ramps up better in multiple-process architectures.

Multi-Thread Utilization Overview

For the reasons set out above, Convertigo recommends deploying several medium-sized instances of C-EMS rather than one big instance.

On a standard Intel machine, it is best to deploy 3 or 4 Convertigo instances, each limited to 15 threads, in order to achieve optimal utilization of hardware resources.

NB: This deployment model, optimized from the viewpoint of thread concurrency, is not penalized by excessive memory consumption: modern operating systems allow sharing of the constant memory costs of the JVM and JRE.

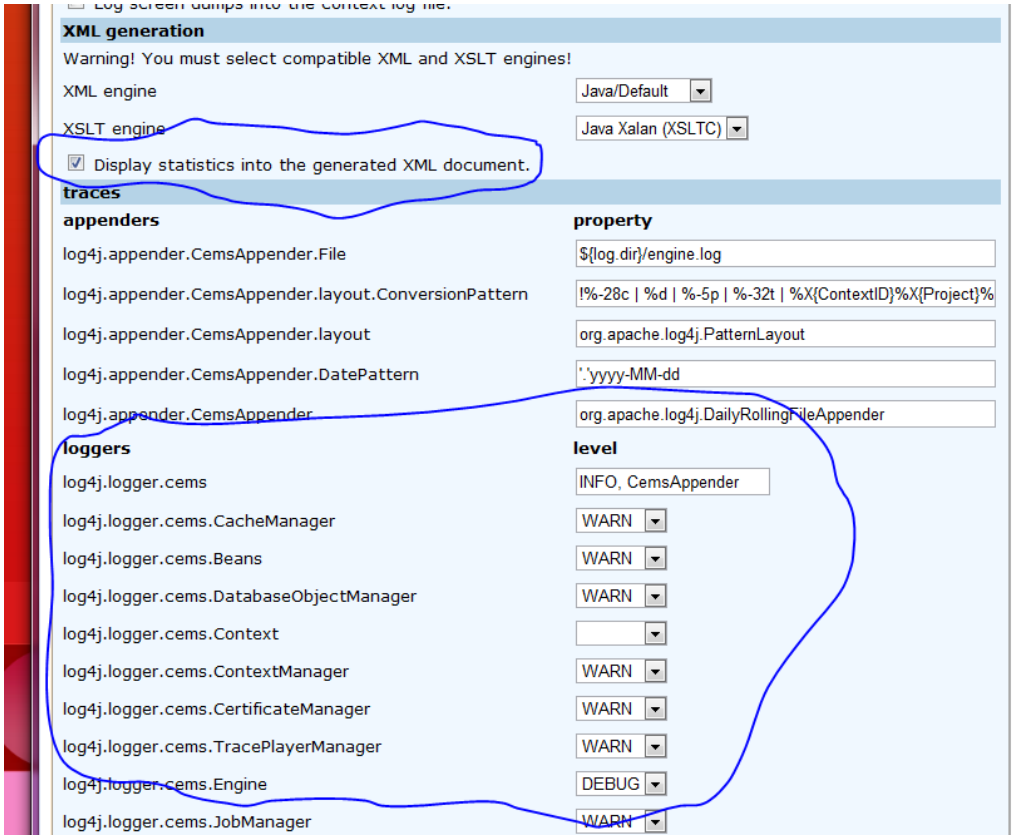
Sequencer

The Convertigo sequencer has a very low impact on overall response time. All operations are performed in memory and involve only a few objects, even for complex sequences. Basically, the execution time of a sequence is the sum of the execution times of the transactions involved in that sequence.

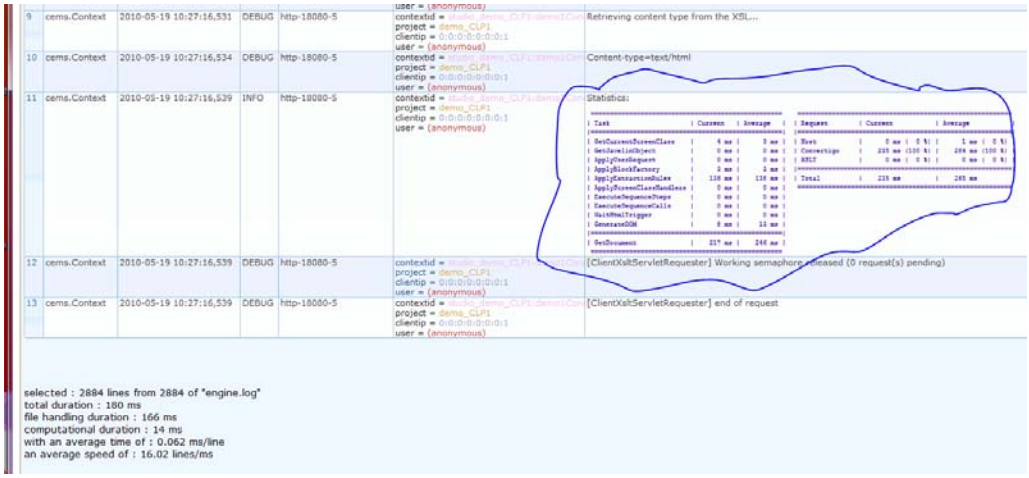
NB: Version 5.0.4 of Convertigo (due for release mid-2010) will bring a significant improvement in sequencer performance by avoiding any unnecessary HTTP network calls.

Metrics

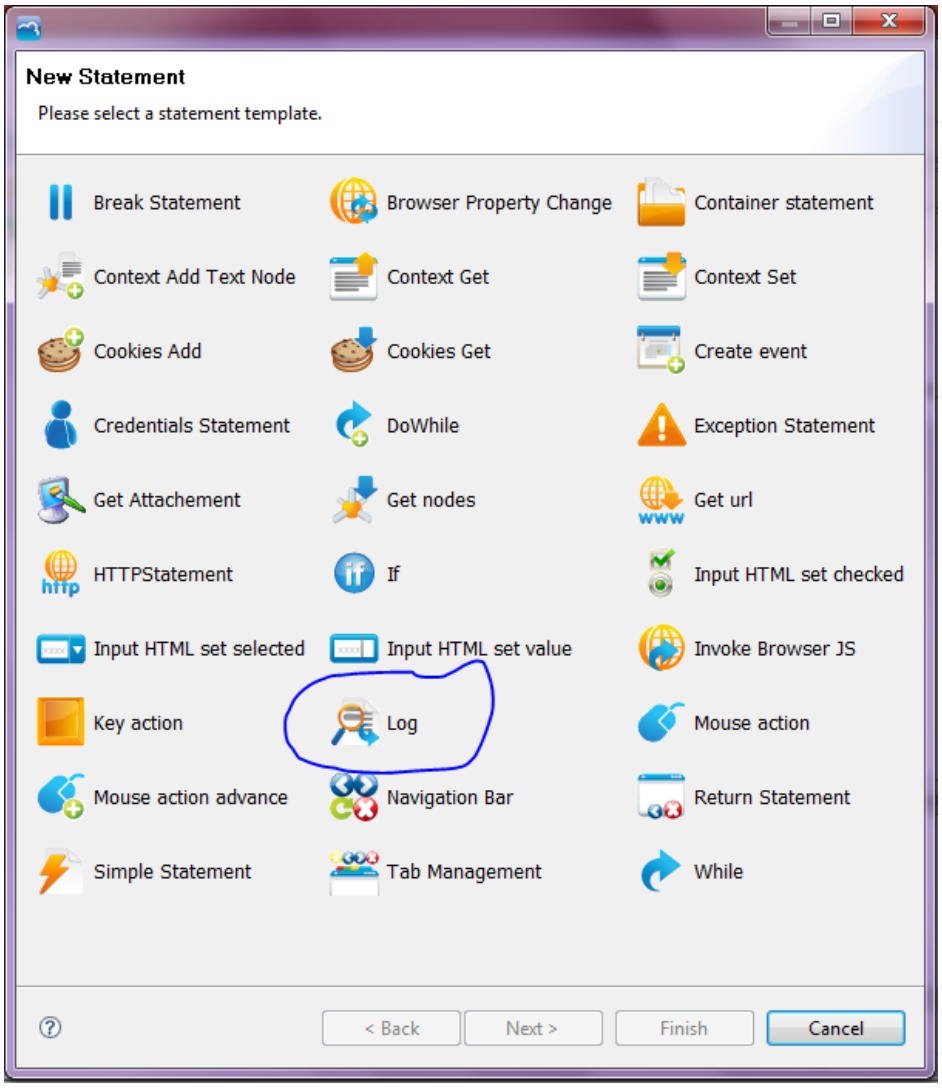
To observe and analyze C-EMS performance, the *Display Statistics* option must be enabled in the Convertigo administration console:

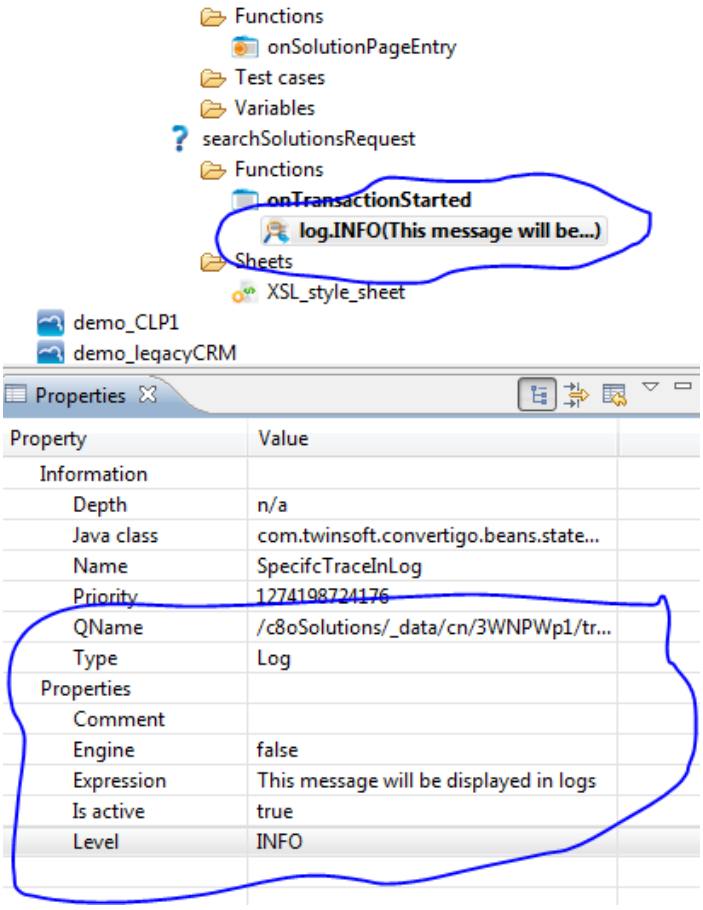


With these settings, the traces provided in the Convertigo engine log files and in the XML outputs will give the response times for each step in the execution of the transactions:



Alternatively, specific traces can be written during the execution of the transactions, using log statements:





The corresponding traces generated in the log files will contain time stamps, which are useful for detailed analysis of response times from a section of a particular transaction.

Performance Optimization

Parallel Steps

The Convertigo sequencer can execute several transactions in parallel (ParallelStep). Under the right conditions – notably when the legacy application response times are long – this means that the execution time for *n* transactions can be limited to the longest transaction time, instead of adding together a series of execution times, as when the transactions are executed by a user.

Pooling

Pooling is a way of pre-initializing connections to existing applications, by executing the login and transaction phases required to enter the application's operational state. Once the pool has been initialized (by the "warming" phase), the next requests will make use of the already active sessions, thus avoiding the often quite significant connection times.

There are two pooling modes:

- A stateless mode, in which the pool entry returns to its initial state after each use;
- A stateful mode, in which the entry stays in its current state, thus retaining a complete context between consecutive calls, before ultimately returning to its initial state.

NB: C-EMS can detect when a pool entry has not returned to its initial active state. If so, it regenerates the entry by closing and re-opening it.

In C-EMS version 5.0, the pooling mechanism is available only for the Legacy Integrator and SQL connectors. Pooling support for the Web Integrator connector is scheduled for the end of 2010.

Enabling pooling can produce a huge improvement in the performance of legacy mashups, depending on the architecture of the original application.

Cache

C-EMS has its own integrated cache mechanism.

Each transaction or sequence can retrieve its data from the cache instead of calling the legacy application.

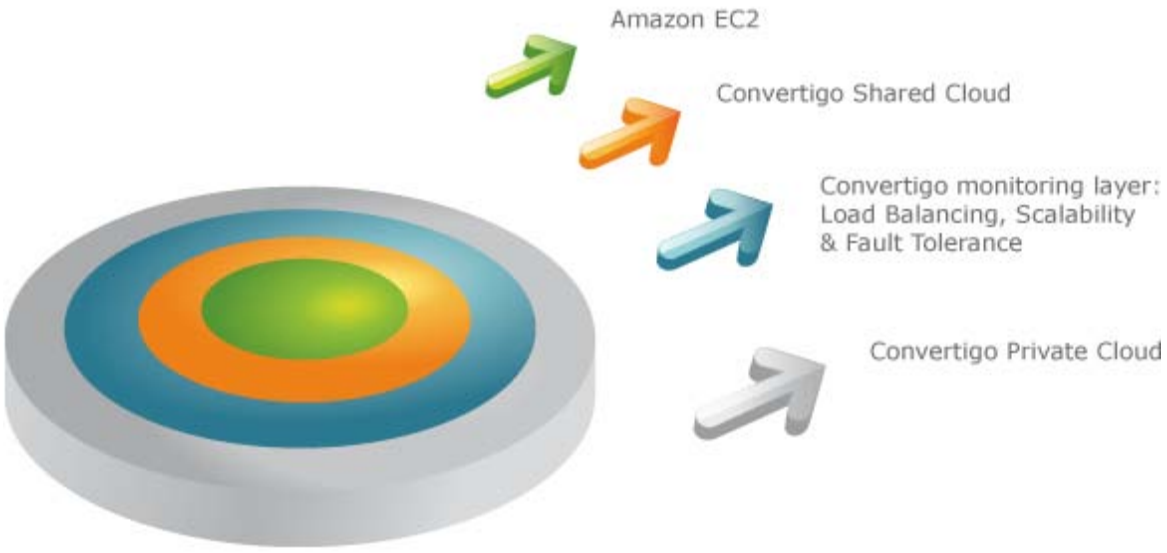
The cache options are configured in the transaction and sequence properties:

- Cache validity period: time (ms), daily, monthly, annual
- Cache entry ID: selection of parameters to be included in the identifier
- Cache location: local XML file or SQL database.

If the cache needs to be pre-loaded, a specific sequence will need to be developed, which will call the various target transactions in order to load the cache.

Cloud

As from version 5.0, C-EMS projects can be deployed in the Convertigo Cloud. The Convertigo Cloud is based on Amazon EC2, supplemented by monitoring and load-balancing tools developed by Convertigo.

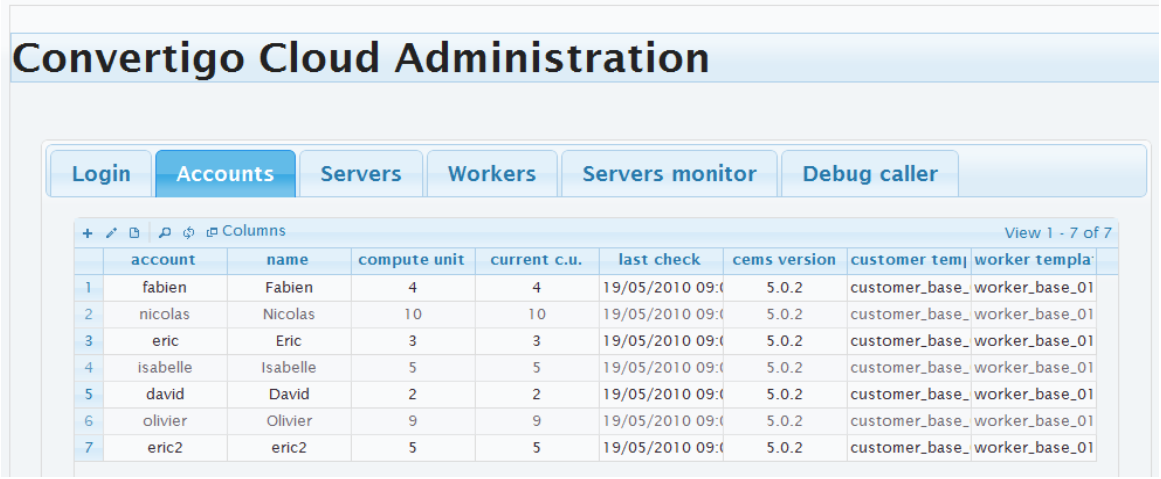


Deployment in the Cloud is done entirely transparently, requiring no Amazon EC2 competencies. Cloud deployments can be combined with conventional server deployments hosted on-premises and administered by customers.

The Convertigo Cloud offers numerous advantages for customers at every phase of a project. In particular, it makes the technology evaluation phases very straightforward.

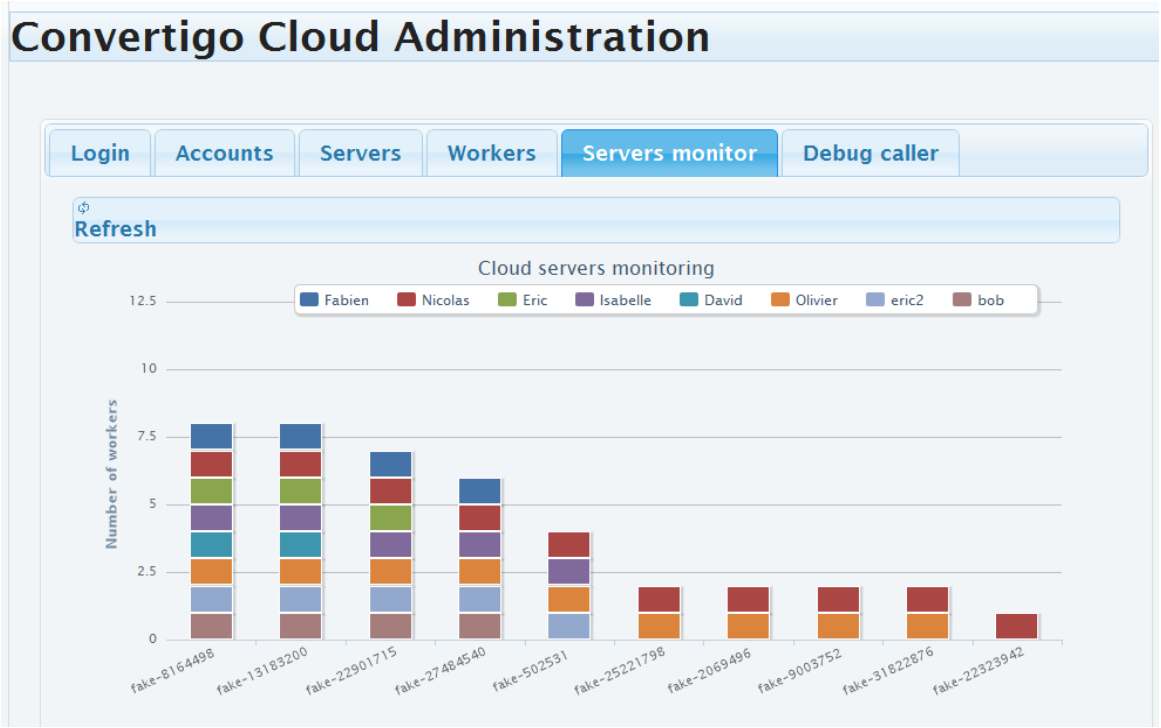
From a production performance viewpoint, the Convertigo Cloud offers total flexibility: Convertigo administrators can rapidly allocate new resources to a project at any time. Conversely, if a project has more resources than it needs, they can again be adjusted very quickly.

Performance-wise, in comparable contexts, Cloud-based execution is similar in terms of response times to running processes on enterprise servers.



	account	name	compute unit	current c.u.	last check	cems version	customer tem	worker templa
1	fabien	Fabien	4	4	19/05/2010 09:(	5.0.2	customer_base_	worker_base_01
2	nicolas	Nicolas	10	10	19/05/2010 09:(	5.0.2	customer_base_	worker_base_01
3	eric	Eric	3	3	19/05/2010 09:(	5.0.2	customer_base_	worker_base_01
4	isabelle	Isabelle	5	5	19/05/2010 09:(	5.0.2	customer_base_	worker_base_01
5	david	David	2	2	19/05/2010 09:(	5.0.2	customer_base_	worker_base_01
6	olivier	Olivier	9	9	19/05/2010 09:(	5.0.2	customer_base_	worker_base_01
7	eric2	eric2	5	5	19/05/2010 09:(	5.0.2	customer_base_	worker_base_01

The Convertigo Cloud automatically adjusts and allocates resources in response to client needs, and in order to maximize the shared use of available Amazon resources (Convertigo load-balancing algorithms).



For fault tolerance reasons, each client is guaranteed to run on at least 2 different Amazon AMIs. The Convertigo Cloud monitoring system detects when an AMI is down and restarts it automatically.

Sizing

Convertigo Overhead

By referring to the *Web Transactions* and *Legacy Transactions* sections, one can calculate the standard Convertigo overheads per type of application, for a standard Intel machine (as defined earlier).

Web Transactions

Minimum processing time: approx. 45 ms, without the optional steps (server-side XSLT, billing).

Main parameters affecting the overhead:

- Screen complexity (affects the parsing step):
  - Very simple: no JavaScript, short page, few tag embedding levels, etc.
  - Average: some JavaScript, tables, e.g. a Google search result page
  - Complex: very full page, containing lots of JavaScript, and embedded tables.
- Extraction rule complexity
  - Simple: a single rule, easy to detect and to execute, producing a short result
  - Average: a rule producing a potentially long result, with loops on several result pages, but without transformations or complex structuring
  - Complex: several rules, producing voluminous results, with costly formatting.

	Simple extraction	Average extraction	Complex extraction
Simple screen	80 ms	160 ms	1060 ms
Average screen	115 ms	**200 ms**	1100 ms
Complex screen	165 ms	245 ms	1145 ms

Legacy Transactions

Minimum processing time: approx. 30 ms, without the optional steps.

	Simple extraction	Average extraction	Complex extraction
Simple screen	40 ms	45 ms	135 ms
Average screen	45 ms	**50 ms**	140 ms
Complex screen	50 ms	60 ms	150 ms

CPU Consumption

On a standard Intel machine (as defined earlier) and with transactions of average complexity (the mean/mean value in the above tables), the estimated processing capacity is about 5 Convertigo Web transactions or 20 Convertigo legacy transactions per second and per processor core, i.e. about 8 Web transactions or 34 legacy transactions on a computer with a dual-core processor.

A “transaction” corresponds to a function in Convertigo Studio and includes a start page / screen, multiple actions on that page (filling in certain fields, clicking buttons, etc.), then receiving and processing a response page (identifying the screen class, executing the various extraction rules) and putting together the result.

Note: When calculating the number of transactions one should take into account the number of legacy application screens / pages that are actually accessed. For example, if the execution of a transaction requires a prior connection, then two screens will have been accessed.

As we saw in the *Simultaneous Transactions* section, you should avoid running Convertigo servers near the saturation limit of their CPU and memory resources. In order to preserve headroom for absorbing peak loads, we suggest a resource utilization value of 50%, allowing for 4 Web transactions or 17 legacy transactions.

4 Web transactions per second =

- >100,000 transactions a day (8 hrs/day) or
- >300,000 transactions a day (24 hrs/day).

17 legacy transactions per second =

- >450,000 transactions a day (8 hrs/day) or
- >1,250,000 transactions a day (24 hrs/day).

If you need to execute a larger number of transactions, you should either use a more powerful machine (faster, more processors, etc.) or deploy multiple C-EMS servers.

Memory Consumption

A Convertigo server takes up at least 256 MB in a JVM.

The memory footprint of a Convertigo legacy context is about 7 MB, allowing some 300 contexts to be handled simultaneously in a well-configured JVM, in a 32-bit environment.

The memory footprint of a Convertigo Web context is about 10 MB.

To these quantities, you should add the space for the connection pool and the cache, if used.

Sizing process

All of the above values are averages, which may be useful at the start of a project. But before any deployment phase, you should enable the traces in the C-EMS administration console and take real measurements of the various steps in your actual Convertigo transactions.

To begin with, these measurements should be made without enabling the different Convertigo optimization mechanisms. Based on these response time observations, one can then determine which Convertigo optimizations are the most relevant, verify that the response times evolve as expected, and size the servers accordingly.

Conclusion

To be able to size Convertigo servers correctly in production, you need to have a solid understanding of how C-EMS works. Although the Convertigo consultants are available to help the deployment teams calculate or validate sizings on request, it is perfectly possible to do the necessary calculations in-house.

It is important to understand that C-EMS calls existing applications at an external level, just as users would. This preserves all of the existing business logic, without the need for new development, but it also introduces an intermediate layer, and therefore an additional response time. This additional time is generally negligible compared to the execution time of the legacy applications, but it is important to factor it in when calculating how many simultaneous transactions a given server can handle.

Remember that it is preferable to deploy multiple C-EMS instances of average size on a server rather than a single very large instance.

The C-EMS Cloud deployment solution provides appreciable flexibility for dealing with sizing questions.

Disclaimer: the numerical data (response times and transaction numbers) used in this document represent values observed in a specific hardware and software environment, on a particular type of application, and should only be understood as indicating orders of magnitude.

On no account do they represent a commitment by Convertigo with regard to the performance and optimization of applications produced with C-EMS.