An Example in Remote Computing Over the Internet applied to Geometry

Ferreira, M[1] and Casquilho, M.[2]
[1] *Department of Computer Science and Engineering,*
Instituto Superior Técnico, Technical University of Lisbon, Lisboa, Portugal
[2] *Centre for Chemical Processes, Department of Chemical Engineering,*
Instituto Superior Técnico, Technical University of Lisbon, Lisboa, Portugal

***Abstract* — Scientific computing over the Internet can suit many activities that have not, in the authors' opinion, been explored enough in general. Resources such as executables, languages, packages, can be used from a remote computing system. In this study, largely based on academic practice, a simple illustrative example in Geometry is implemented on a distributed system that outsources the computing-intensive tasks to remote servers that may be located in other universities or companies, linked to grids and clusters and so on. The software stack and software developed to support the communication is explained in detail. The architecture developed stresses the interoperability of the software, and a suitable high degree of decoupling between components hosted in various locations. The results of this study motivate further work and serve a practical purpose that may be useful to everyone doing scientific computing.**

***Keywords* — Internet, remote executables, Scientific computing, university-industry links.**

I. INTRODUCTION

MANY areas of scientific computing can be addressed over the Internet, but this approach has not, in general — in these authors' opinion — been appropriately explored, all the more if compared with most uses of that ubiquitous communication network. One of the authors has, since more than a decade, intensively used this mode of computing in research and teaching at his university work, in domains related to Mathematics, namely Operational Research, Statistics or Chemical Engineering. The computing has been mainly done in a server of the university's information technology centre, intended typically to host faculty and students' webpages. The present study, largely based on that previous academic practice, focuses on the establishment of a link between two universities, one supposedly wishing to execute software made available by the other. This would also apply to any two entities, such as a set of two companies or a university-company linkage (a particular application of [5]). In the Internet context, resources adequate to the particular technical purpose, such as executables, languages, or packages, can be used, if accessible at this level with due permissions, from a remote computing system.

The Internet affords nowadays an unprecedented ease of communication at a very low cost, so that a step can be taken to reap benefits from using remote resources. There are, of course, many resources for computing on the Web, dealing with small tasks, ranging from conversions of units to more complex mathematical problems. Regarding scientific computing over the Web, an extensive example of this activity in the academic environment is the original work by Ponce ([7]), containing a large number of (Fortran) programs to solve problems dealing with Hydraulics and related areas in Civil Engineering. These applications are presumably (as all of our previous work) deployed wholly on single nodes, which also host the web interface and logic. Building on such projects as the excellent one referred above and our own previous projects, the present work intends to take this topology into a next stage, allowing further decoupling of components, by introducing an intermediate communication layer between distributed nodes, which together form the web computing system.

Internet-based computing as an everyday activity has been deemed by one of the authors indispensable to his activities as a tool in the academic practice, and a gateway to the university-industry linkage — widely praised but often scanty — in an era of cheap information technology gear.

The present study is based on a simple, yet surprising, illustrative example in Geometry — an example that might be used in a lecture — chosen both to be clear to a wide readership and to avoid beclouding the underlying software structure. Thus: the problem is started in a webpage of one entity; and the computation is done, without the user's perception, at another machine (suggesting the extension to more), allowing a certain software to be accessed.

In the following sections: the illustrative example is briefly described in its mathematical aspects; the developed resolution based on network computing is presented, with the implemented software architecture to support it; and finally some conclusions are drawn about the proposed solution and system developed.

II. ILLUSTRATIVE EXAMPLE

A problem in Geometry, otherwise conceived as a simple template for more applied cases, was chosen as an illustrative

DOI: 10.9781/ijimai.2013.235

example for the technique. Let the minimum distance be sought between points *A*, source, and *B*, destination, as seen in Fig. 1, both on the *X*-axis, passing by point *P*, to be determined, on the half line *s* making an angle γ with the horizontal axis. The problem is treated in [1] and solved by simple differential calculus. The analytical solution for $P = (X, Y)$ is given in (1).

$$\frac{1}{X} = \frac{1}{2}\left(\frac{1}{x_1} + \frac{1}{x_2}\right)\sec^2\gamma \qquad (1)$$

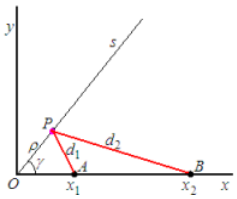

Fig. 1. Route from A to B, passing by P on s, for minimum distance.

With $Y = X\tan\gamma$, (1) leads to (2):

$$X = 2\frac{x_1 x_2}{x_1 + x_2}\cos^2\gamma$$
$$Y = 2\frac{x_1 x_2}{x_1 + x_2}\sin(2\gamma) \qquad (2)$$

More concisely, in polar coordinates, (ρ, θ), with $\theta \equiv \gamma$, the radial coordinate is

$$\rho = X/\cos\gamma = \left(2\frac{x_1 x_2}{x_1 + x_2}\cos^2\gamma\right)\Big/\cos\gamma = \\ = 2\frac{x_1 x_2}{x_1 + x_2}\cos\gamma \qquad (3)$$

The interest of this problem — the reason it was chosen — lies in the unexpected result as γ decreases towards 0. In Fig. 2, the optimum routes are shown, to which correspond the optimum positions of *P*, for various descending values of γ, always with $x_1 = 1$ and $x_2 = 3$. The results come from the authors' website ([2]). Now, intuition would possibly lead ρ to the *arithmetic mean* of x_1 and x_2.

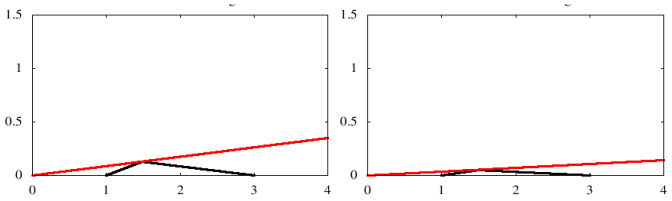

Fig. 2. Optimum routes for = 60, 40, 20, 10° (from left to right).

Observation of the sequence in Fig. 2, however, disputes intuition, and confirms (3): ρ tends to the *harmonic* mean of x_1 and x_2. Images for small angles, 5 and 2°, in Fig. 3, show the limiting ρ to be not 2, the arithmetic mean of $x_1 = 1$ and $x_2 = 3$, but 1.5, their harmonic mean.

The adequacy of the arithmetic mean in its own right should be noted (for $\gamma = 0$), notwithstanding, by verifying that, just by letting x_1 and x_2 grow indefinitely, with $x_2 - x_1 = \delta$ (δ constant), the harmonic mean tends to the arithmetic mean, as seen in (4).

$$\lim_{x_1\to\infty}\frac{1}{\rho} = \frac{1}{2}\left(\frac{1}{x_1} + \frac{1}{x_1 + \delta}\right) = \\ = \frac{1}{2}\frac{x_1 + \delta + x_1}{x_1(x_1 + \delta)} = \frac{1 + \frac{\delta/x_1}{2}}{x_1 + \delta} = \\ = \frac{1}{\frac{x_1 + \delta}{1 + \frac{\delta/x_1}{2}}} \qquad (4)$$

Considering the infinitesimal δ/x_1, (4) becomes (5), where the arithmetic mean is now visible ($\rho = x_1 + \delta/2$).

$$\lim_{x_1\to\infty}\frac{1}{\rho} = \frac{1}{(x_1 + \delta)\left(1 - \frac{\delta}{2x_1}\right)} = \\ = \frac{1}{x_1 + \delta - \frac{x_1\delta}{2x_1} - \frac{\delta^2}{2x_1}} = \frac{1}{x_1 + \frac{\delta}{2}} \qquad (5)$$

Another interesting property of the optimum routes is that, for varying γ (with fixed x_1, x_2), the locus of the optimum points *P* is a circle with radius $R = x_1 x_2/(x_1 + x_2)$ (same physical units of the *x*'s, of course) centred at (0, *R*), here $R = 3/4$. These facts, out of the scope of this study, corroborate the adequacy of the Internet also to openly reveal noteworthy features.

III. Software Architecture

This study is based on previous applications for many types of scientific problems and expands their capacity using the Internet, following past and current academic practice. In this work, we developed a decentralized computing architecture, distributed on a network, using the HTTP protocol to communicate between the servers, in what is usually known as a web4 service. The architecture is composed by servers playing two separate roles:

a) a front end role, providing the computing services to the clients, with a simple, practical web interface that can be easily accessed through any browser; and

[4] In *web* (as attributive) or *Web*, the Chicago Manual of Style Online ([4]) was roughly followed.

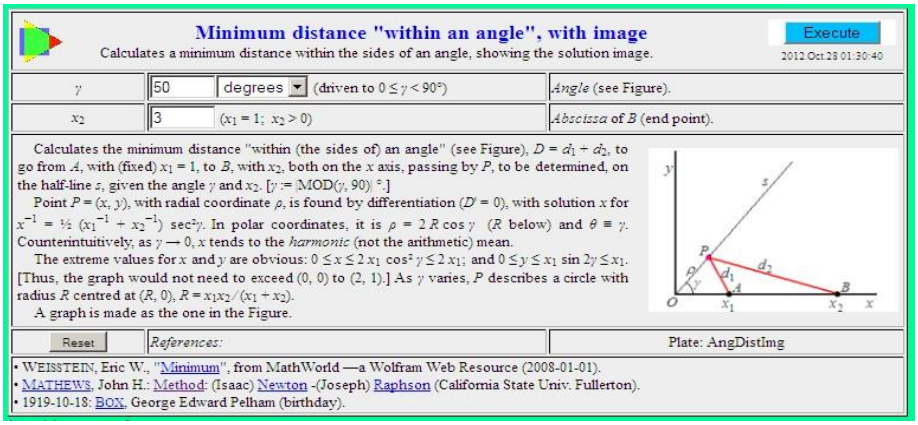

Fig. 3. Optimum routes for = 5, 2° (from left to right).

b) a back end role, receiving the computing tasks from the front end and having the required software to execute them.

The remote call may incur a substantial delay, depending only on the network latency, and mainly the complexity of the problem and computing power of the remote server machine.

The back end addresses must be known to the front end servers, so that they can be located on the public network. Likewise, the front end must be publicly accessible to the users/clients, and have a well-known address.

In the architectural layout described, both the front and back end servers are highly decoupled between them and from the other servers, having no structural dependencies on any single network point [no SPOF5 (e.g., [4])]. Therefore, they can be easily brought up and down, and change location, without disturbing the overall functioning of the system, which grants a very valuable comparative advantage. The only requirement for the system to work is just one front end and one back end servers online at any given time.

The decoupling is highly beneficial for two reasons: i) load balancing of requests between the front ends, and of computing tasks between the back ends; and ii) fault tolerance against possible node crashes.

The front end and the back end support parallel task/requests that require a separation and isolation of execution contexts. This is guaranteed by the HTTP server and the script engine used, which is PHP, with additional safeguards required in the code to carefully avoid any conflict in the resources used (filenames, etc.).

The system is illustrated in this study with the geometric example above, implemented on an Internet link between two semi-closed local networks, the Sigma cluster of IST, and the web servers of FCUL, following the steps described in the next two subsections.

The IST server is deployed on a cluster of AMD64 Opteron processors (2.4 GHz) running Debian Linux, Apache 2.2.16, and PHP 5.3.3-7. The FCUL server runs on a cluster of i386 Intel Xeon processors with Red Hat Linux, Apache 2.2.3 and PHP 5.1.6.

[5] "single point of failure"

Local execution

The starting point of the study, based on previous work done, was a system deployed in a single local server. This system combines the front end and back end functionalities locally. This is a simple case scenario that served to develop and test the basic computing service.

The system uses the following five files in turn:

a) Webpage, such as [2], in a well-known address of a front end server — It is a PHP file containing an HTML 'form' to receive the user's data, which is then sent via an HTTP POST method to a processing PHP script (following item);

b) PHP script 'interface.php', which

1. Extracts the user's arguments from the HTTP request;
2. Launches the required program in a new process (via PHP's 'proc_open') with redirected streams to new process pipes, open to the calling PHP process;
3. Feeds it with the given arguments through the child process read pipe;
4. Waits to read the output of the called program from the other, write pipe; and
5. Closes the pipes and terminates the child process.

c) Binary program ('angDist.exe', compiled from a Fortran 90 source), which also writes to the output stream the data required for a graphic to be created afterwards.

Now, the 'interface.php' script [in b)] constructs a dynamic webpage from:

a) 'interfacetop.php' (constant), the top of the webpage;

b) body (main) section, in HTML 'pre' format, with the results of the program call, and (typically) a graphic with plotted results, closing HTML bottom.

The screenshots are shown in Fig. 4 for the user data and in Fig. 5 for the results of the computation.

Results

```
2012-10-28 UTC+0000  1:36:55.952
*** Distance within an angle ***                              (Oct-2012, MC)

gamma (deg., rad),  50.0      0.8727    |
x_1, x_2,             1.       3.000    |
-------------------------------------------------------------------------------
COS, TAN(gamma), 0.64279      1.1918    | COS^2,                    0.41318
                                        |
If x_2 = x_1:                           | (absolute minimum, d0)
  x_P, y_P,      0.41318      0.49240    | d0 = x_1 cos^2 = x_P =    0.41318
If x2 = infinity:                       |
  x_P, y_P,      0.82635      0.98481    | x_P = 2 d0
                                        |
MINIMUM DISTANCE for given x_2          |
x_P, y_P,        0.61976     0.73861    | Cartesian coordinates
rho_P,                       0.96418    | (Polar) radial coordinate
d_1, d_2,         0.8307       2.492    | A--P, P--B
Min. distance,                3.3229    | A--P--B

2012-10-28 UTC+0000  1:36:55.952
2012-10-28 UTC+0000  1:36:56.386        CPU:       0.4 sec.          End
```

Fig 4.. Webpage for the user data..

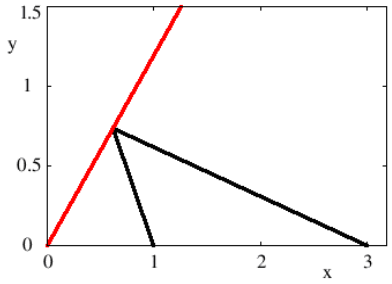

Fig. 5. Webpage for the results.

A. Remote execution

The remote execution mode is the focus of the present work. In this mode, the computing component was distributed to a remote network allowing for a scalable expansion of the system by adding more computing nodes. The decoupling adopted thus requires the development of a middle-ware communication layer between the web-interface (front-end) and the computing nodes (back-end). The decoupled architecture also provides scalability for the front-end, allowing the deployment of multiple interface nodes, scaling up according to the number of incoming requests. The accessible web front-end is available in [3]. The system can be easily deployed throughout many nodes, which can be switched on and off depending on the desired system throughput and efficiency.

Starting from the local execution system described in the previous section, the interface between the computing program and the web front end was greatly modified to support the distribution of both parts, mainly the computing intensive tasks. A muddle-ware was developed to implement the network communication, with the required transfers and conversions of data. The process of service lookup by the remote servers is done by a semi-static approach, i.e., a list of hostnames of the known service providers, contacted in sequence until a live one replies.

To the desired end, the following changes were made:

a) Refactoring the PHP complete service module, into two separate modules: a local front end component, and a back end web service interface for the remote program;

b) The front end interface loads the list of known back end servers' addresses, and polls them to find one available;

c) The front end makes an HTTP request to the available server, by invoking the PHP script on the back end. The front end forwards the input data using the HTTP POST method, specifying in the request which service is required (i.e., 'angDist' in the example);

d) The back end interface calls the binary program in a manner similar to the local execution mode, executing the requested task in isolation in that node;

e) The back end sends the results back to the front end, i.e., both the main results and the parameters of the to-be-created graphic, formatted following a well-defined template, and packaged in the same HTTP response body;

f) The front end process receives the output of the task, and unpacks the two blocks of data (results and graphic's parameters), which have been pre-formatted accordingly; and

g) The front end retains the responsibility of generating the graphic with the parameters received from the remote request, using the GNU tool gnuplot.

The choice was made not to send the graphic itself over the Web, for it could lead to problems of text data encoding (one of the tenets of web services being the use of textual ASCII data), and it would considerably increase the messages' payload size.

The results for the user are, of course, the same as previously. A different HTML background image was chosen to differentiate between a service running in local execution mode (the front end at IST) and another in remote mode (the one at FCUL [3]). The remote execution network is schematically shown in Fig. 6.

The system performed as expected, namely, the communication latency introduced by the network was negligible when compared to the typical computing time for scientific problems, and it is a constant delay depending only on the size of input and output data, and the underlying network infrastructure.

The remote execution mode is the focus of the present work. In this mode, the computing component was distributed to a remote network allowing for a scalable expansion of the system by adding more computing nodes. The decoupling adopted thus requires the development of a middle-ware communication layer between the web-interface (front-end) and the computing nodes (back-end). The decoupled architecture also provides scalability for the front-end, allowing the deployment of multiple interface nodes, scaling up according to the number of incoming requests. The accessible web front-end is available in [3]. The system can be easily deployed throughout many nodes, which can be

switched on and off depending on the desired system throughput and efficiency.

Starting from the local execution system described in the previous section, the interface between the computing program and the web front end was greatly modified to support the distribution of both parts, mainly the computing intensive tasks. A muddle-ware was developed to implement the network communication, with the required transfers and conversions of data. The process of service lookup by the remote servers is done by a semi-static approach, i.e., a list of hostnames of the known service providers, contacted in sequence until a live one replies.

To the desired end, the following changes were made:

a) Refactoring the PHP complete service module, into two separate modules: a local front end component, and a back end web service interface for the remote program;

b) The front end interface loads the list of known back end servers' addresses, and polls them to find one available;

c) The front end makes an HTTP request to the available server, by invoking the PHP script on the back end. The front end forwards the input data using the HTTP POST method, specifying in the request which service is required (i.e., 'angDist' in the example);

d) The back end interface calls the binary program in a manner similar to the local execution mode, executing the requested task in isolation in that node;

e) The back end sends the results back to the front end, i.e., both the main results and the parameters of the to-be-created graphic, formatted following a well-defined template, and packaged in the same HTTP response body;

f) The front end process receives the output of the task, and unpacks the two blocks of data (results and graphic's parameters), which have been pre-formatted accordingly; and

g) The front end retains the responsibility of generating the graphic with the parameters received from the remote request, using the GNU tool gnuplot.

The choice was made not to send the graphic itself over the Web, for it could lead to problems of text data encoding (one of the tenets of web services being the use of textual ASCII data), and it would considerably increase the messages' payload size.

The results for the user are, of course, the same as previously. A different HTML background image was chosen to differentiate between a service running in local execution mode (the front end at IST) and another in remote mode (the one at FCUL [3]). The remote execution network is schematically shown in Fig. 6.

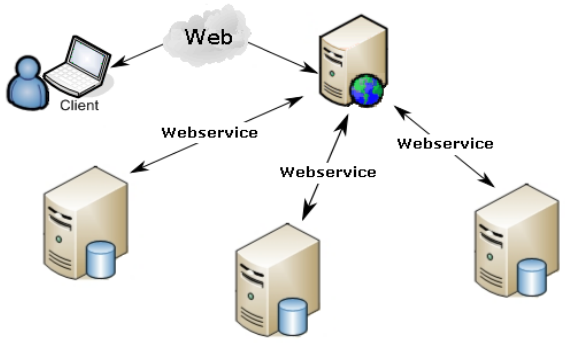

Fig. 6. Remote execution network.

The system performed as expected, namely, the communication latency introduced by the network was negligible when compared to the typical computing time for scientific problems, and it is a constant delay depending only on the size of input and output data, and the underlying network infrastructure.

IV. Conclusions

The present study inherits former extensive work in scientific computing over the Internet by one of the authors, akin to the work by [7]. Our work has been done in one server of IST, where the webpages and their respective executables are located. The study extrapolates that approach to a two-server solution permitting a webpage on a new server, at FCUL, to access an executable placed on the other server, at IST, without the user's perception. The access is governed by two PHP scripts, each placed in one of the servers.

This shows the ease of use of an executable in a remote locus possessing required resources (executables, languages, packages), thus avoiding the breach of the source webpages' style. With the current ease of communication, this points to the use of remote software among collaborating entities, such as companies or universities or in the university-industry linkages. Thus, some software components topologically isolated from a web gateway or from unsecure locations outside its LAN may be accessed by a trusted web server and provided to the worldwide web users.

Acknowledgment

The research was done at (1.st author) the Department of Computer Science and Engineering and (2.nd author) Centro de Processos Químicos (Centre for Chemical Processes), Department of Chemical Engineering, both of the Technical University of Lisbon, Lisbon, Portugal. Thanks are due to the CIIST (Centro de Informática, Informatics Centre) of Instituto Superior Técnico and, for their special effort, CI of Faculdade de Ciências (Faculty of Sciences) of the Lisbon University.

References

[1] Casquilho, M., Buescu, J.: A minimum distance: arithmetic and harmonic means in a geometric dispute, International J. of Mathematical Education in Science and Technology, 147, 399–405 (2011).

[2] Ferreira, M., http://web.ist.utl.pt/ist11038/compute/com/Fx-angdistImg.php

[3] Ferreura, M., http://webpages.fc.ul.pt/~maxxxxxxxxx/compute/Fx-angdistRemote.php
[4] Chicago Manual of Style Online (The), http://www.chicagomanualofstyle.org/
[5] Dikaiakos, M. D., Katsaros, D., Mehra, P., Pallis, G., Vakali, A.: Cloud computing: distributed internet computing for IT and scientific research, Internet Computing, IEEE, 13(5), 10–13 (2009).
[6] Dooley, K.: Designing large-scale LAN's. O'Reilly Media, Inc., Sebastopol, Ca. (USA) (2002).
[7] Ponce, V. M.: Vlab, http://onlinecalc.sdsu.edu/, San Diego State University.

Miguel Ferreira was born in Lisbon, Portugal, in 1979. He got his M. Sc. in Computer Engineering from Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, in 2013.

Miguel Casquilho was born in Lisbon, Portugal, in 1948. He got his M. Sc. from Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, in 1971 and his PhD in the same subject and University. He has been an Assistant Professor till 2012, now retired, and has worked in scientific computing on the Internet in modelling, simulation and optimization in Operations Research, Quality Control, and Computing. Prof. Dr.Casquilho is a member of the Union of Portuguese Engineers, and of the American Society for Quality.