Unrestricted VCC Development Framework for High-Throughput Applications in Computer Vision

Alexander Shorin, Habib Naderi, Georgy Gimel'farb, and Patrice Delmas
University of Auckland, Department of Computer Science, P.B. 92019, Auckland 1142, New Zealand
Email: {al,habib,ggim001,pdel016}@cs.auckland.ac.nz

***Abstract*—Virtual computational cloud (VCC) architecture is an attractive option for many researchers in computer vision due to its ability to access and manipulate very large repositories of images remotely. Often an integration of local and remote computational environments and image repositories is required as geographically dispersed collaborating teams do not have unrestricted access to image repositories due to timing, legal, ethical, architectural or bandwidth limitations. An integrated distributed computational framework developed using a VCC architecture allows keeping a synchronised repository of shared code which can be used to access remotely located image repositories. None of the out-of-shelf cloud engines were appropriate for this task simply because of security and configuration limitations they impose. On the contrary, our engine allows unrestricted development and deployment under C++ and Matlab compilers and runtime environments. This paper describes the private VCC framework and its implementation details. The proposed system is easily scalable and can be deployed in other research institutions upon request.**

I. Introduction

Cloud computing is one of the branches of the parallel computing paradigm which offers researchers computing resource and data throughput capability not available otherwise [1], [2]. Cloud computing gained attention of the business and consumer communities, and it offers an attractive opportunity for research in computer vision — a direction which has been scarcely explored so far [3], [4], [5]. The need for an effective, scalable virtual computational cloud (VCC) architecture for research purposes is precipitated by the explosive growth in the amount of binary data being collected, stored and utilised in geographically distributed research institutions across the globe. As collaboration between the teams grows so is their demands for access, computing, and bandwidth allocations.

There are a number of problems which can potentially impede a shared computer vision research project. In medical imaging for example, the access to image repositories maintained by medical schools can be restricted due to ethical or legal considerations. Often in order to participate in a project, developers from other teams have to be physically present within the physical or firewalled environment of the institution, something which is not always feasible. In other situations the sheer size of many medical imaging datasets can be an issue. For example, currently a single compressed CT image set can reach hundreds of megabytes making downloading even a small image repository a challenging task due to bandwidth considerations. Even if repositories could be distributed using courier services, it involves expensive manual process of compiling the distribution media, as well as timing difference involved with delivering it to a remote overseas destination, maintaining it on the second site and so on. Finally, online image repositories might have restricted access times due to a variety of organisational and operational reasons. As bandwidth availability grows over time it will unlikely catch up with ever growing availability of imaging resources.

It would seem that in the situation when a collaborative shared research is undertaken between the teams relying on the same extra large imaging data source a deployment of a cloud infrastructure with high-throughput that could facilitate accessing remote datasets. Unfortunately, in the context of development where languages with low-level features such as C++ or Matlab are used, most of commercially available cloud solutions cannot offer flexible level of security configuration for research developers. Specifically, although deliberately malicious actions by a rogue programmer can probably be excluded as the system is not open to the public, an unintentional error in the code can potentially be devastating for the stability of the entire cloud environment. Addressing security considerations via restricting the development specifications is not feasible as it would render research development on such system impossible.

To the best of our knowledge we are not aware of any unrestricted VCC development framework with low level languages which is commercially offered or otherwise currently available to researchers. In this paper we describe the environment which is currently being developed at the University of Auckland and which successfully overcomes the problems discussed above. The key features of the system is that it is of low maintenance costs, it is easily scalable in terms of number of sites as well as the number of formal languages employed, and that it runs on both Linux and Windows operating systems. Currently, the system supports development in two languages C++ and Matlab. Its pilot version can be accessed by application at http://imaging-frontend.cs.auckland.ac.nz:8080/ImagingJSF.

II. System Description

A. VCC: Defining the Cloud

The system described in this paper is categorized as a VCC because from the standpoint of the user we deliver shared private remotely hosted computational, application, data and storage resources over the public network. The overall system architecture is illustrated in Fig. 1.

The *cloud client* used for service delivery can be any modern browser as described in the documentation available online [6].

Currently, there are two *cloud applications* supported — C++ and Matlab compilers and run-time environments — however, the system can be easily scaled to include other development technologies. The system is deployed both on Ubuntu Linux 10.10 and Windows 7 Professional *cloud platforms* run as VMs, however, because the cloud client is a common browser, the system can be accessed from a compouter running any O/S.

The general top-side architecture of the cloud consists of a singular frontend and an array of backends with the latter being scalable both in terms of cloud platforms supported and also of their geographic or server locations (see Fig. 1).

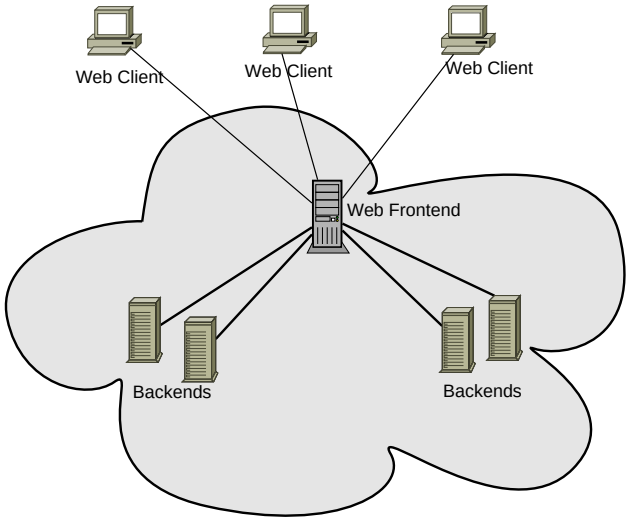

Figure 1. System architecture

The frontend [6] is a web application which runs on a server at the University of Auckland. It is the sole entry point for any cloud client to all services deployed by the system, and since it is a web browser, it does not require any specialized software to access and use the system. From the standpoint of the users, they can only directly interact with the webpages on the frontend, thus hiding backend implementation providing required security, encapsulation and abstraction layering.

The relationship between other components of the system is shown in Fig. 2. Two databases are maintained on the frontend: *image database* which is a central storage location for information about available images in the system, and *code/libraries database* which provides access to stored code and libraries uploaded by the users. The run-time environments installed on the backends have direct access to their respective image repositories which the system views as a set of files in arbitrary formats, organized in the hierarchical structure. The system is platform-scalable by extending the number of backends at each site, and it is site-scalable by cloning the remote cloud deployment sites (left box, Fig. 2).

III. User Interface

Web frontend user interface provides the following facilities for normal users:

- Uploading/compiling code/libraries

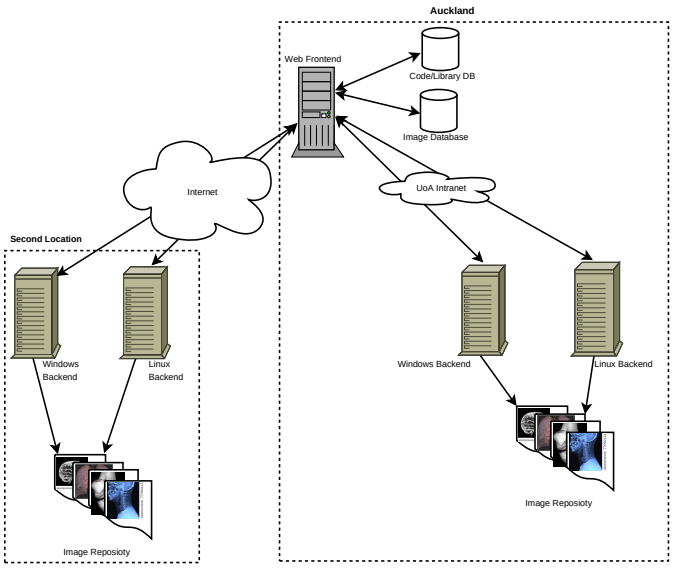

Figure 2. The currently deployed system: both the frontend and two backends are based in Auckland (right), while the second set of backends is based at Bioimaging Lab of University of Louisville, USA (left)

- Browsing code/libraries
- Searching image repositories
- Browsing image repositories
- Running experiments

Extra functionalities are provided for administrators including user administration and traffic/activity monitoring.

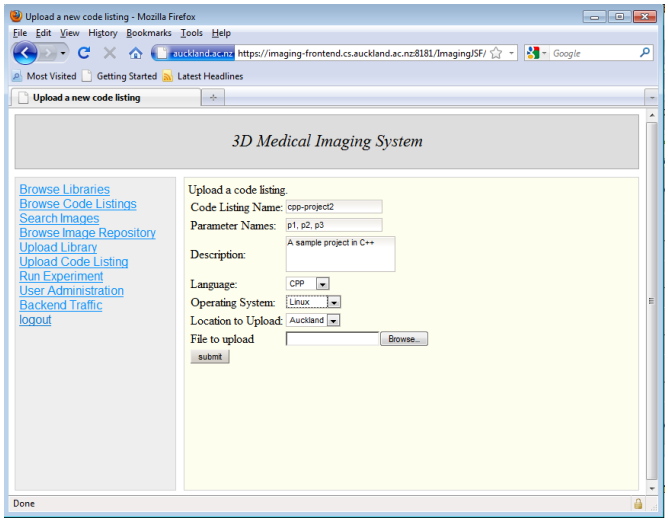

Figure 3. GUI example for uploading source code.

A. Uploading code and libraries

To upload project files and libraries, the user needs to specify the code attributes including name, parameters, description, language, operating system, and the target location (see Fig. 3). The source code is directed to the appropriate backend as per the user choice. Uploadable code is expected to be provided as an archived zip file which is sent to the frontend by the user. The

selected backend uncompresses the `zip` file, compiles source code immediately, and adds the project to the code/libraries database. If compilation errors are detected they are reported to the user and the upload is rejected. Finally, for library uploads the system saves library object files (`dlls` for Windows and `libs` for Linux), while for projects executable files are generated and stored.

The user interaction workflow for uploading new source code is shown in Fig. 8.

B. Browsing uploaded code and libraries

Users are able to see the list of uploaded projects and libraries they own (see Fig. 4). The system has two partitions for project files: temporary and permanent. Temporary projects are automatically removed after 30 days. If the user moves a project to the permanent partition, it remains there until it is deleted by the owner.

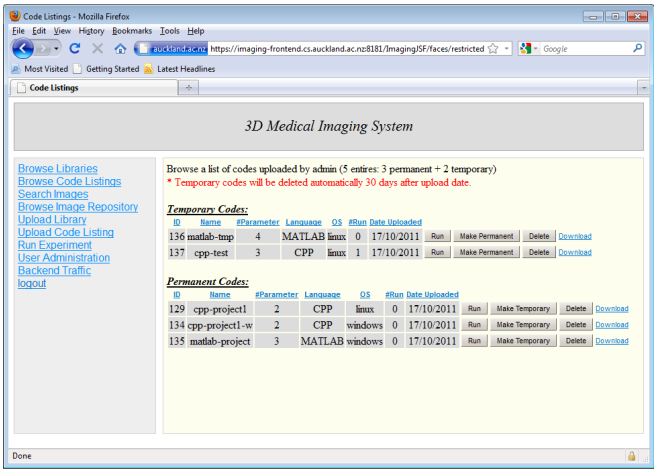

Figure 4. GUI example for browsing uploaded code.

The system allows to link user libraries from any project the user has in development. Libraries are stored in a separate partition and are private to a given account.

Source code needs to be uploaded once only, as the frontend manages code synchronisation across different backends. The frontend also deletes projects and libraries from all locations when required.

C. Searching image database

Users are able to search for an image or a set of images based on available attributes (see Fig. 5). Results of the search is shown to the user as list of selectable entries. Selected images then can be passed as parameters to experiments if required.

D. Browsing image repositories

In addition to searching users are also able to browse and navigate through image repositories. By selecting location and top folder, a user can explore the hierarchy of the selected image repository.

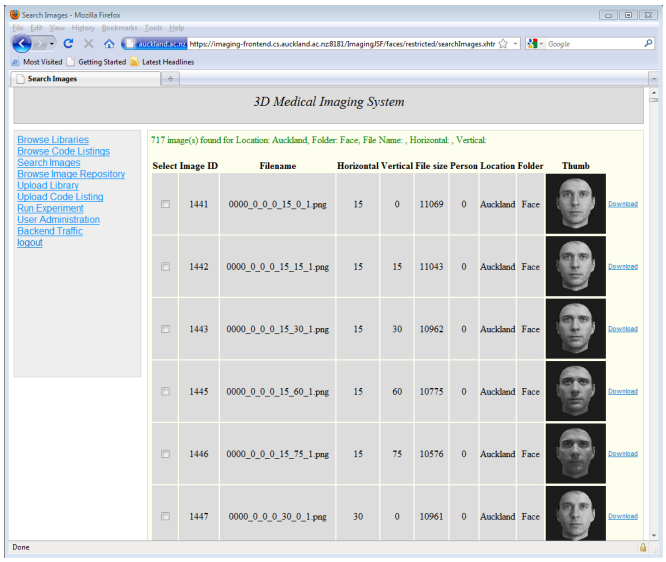

Figure 5. GUI example for searching image repositories.

E. Running experiment

Once project source code is uploaded and compiled, the user can immediately run experiments. The backend creates an output directory and runs the code. If the code runs successfuly, backend sends the address of the output folder to the frontend. Then frontend opens the output folder and shows the content to the user. The user will be able to download individual files or a `zip` file containing all results. Figure 6 shows an example of selection options the user might be offered.

Three options have been provided in the user interface for running experiments:

- **Manual run**: the user manually selects a the project from the list of available code, provides parameters, selects required images, and the relevant OS.
- **Run via command**: by specifying unique code ID of the project, parameters, and image identifiers, the experiments can be run in the "command-line" mode. It is especially useful when the user needs to run near identical experiments repeatedly.
- **Batch run**: a plain text list of commands described above can be uploaded as a text file, and thus enable batch experiment processing.

The user interaction workflow for a manual run is shown in Fig. 9.

F. User administration

This option allows administrators to manage user accounts. Administrators are also able to monitor user activity especially with regard to the system memory and CPU load and backend traffic. Historical data and reports are available to determine any bottlenecks or abuses.

IV. Hardware and Application Architecture

A general description of the architecture is given in Section II-A and Figs. 1–2. Figure 7 shows the detailed software

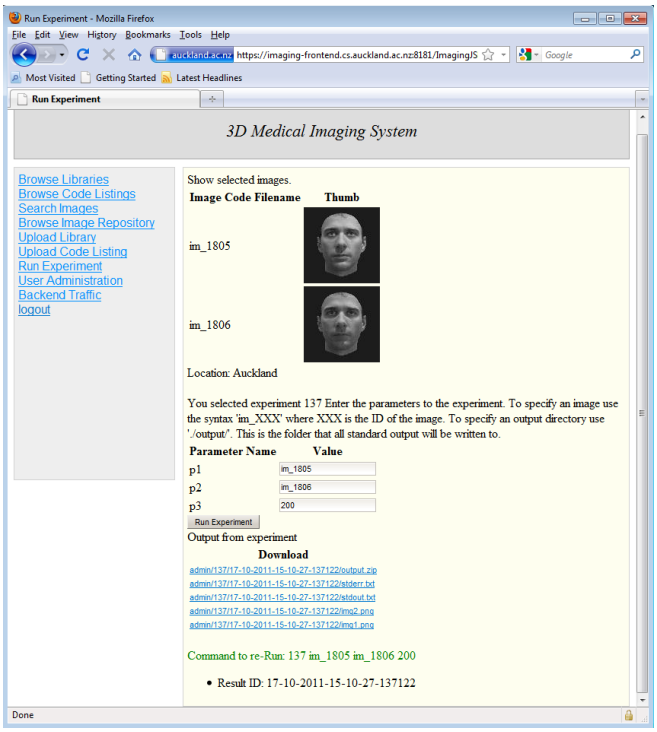

Figure 6. GUI example for running experiments in 'manual run' mode.

architecture implemented in the cloud.

Frontend user insterface has been implemented using JSF (Jave Server Faces) technology. RPC (Remote Procedure Call) has been used to implement a communication channel between frontend and backends. SMB protocol has been employed for transferring files between frontend and backends. Image database and code/library database are managed by mySQL server.

The backend architecture is built upon Windown and Linux VMs, with the compilers and and run time environments for `gcc`, VS C++ and Matlab technologies. The backend also provides the file mount to the actual image repositories which are expected to reside on the same network.

V. Conclusion

This paper describes the architecture and implementation details of the currently developed private VCC framework system for high-throughput applications in computer vision. The key features of the system is that it is of low maintenance costs, it is easily scalable in terms of number of sites as well as the number of formal languages employed, it supports both Linux and Windows operating system environments, and it supports development in two languages, C++ and Matlab.

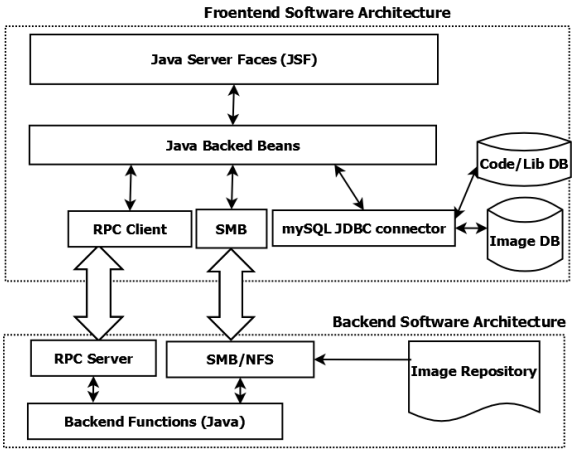

Figure 7. Frontend and backend application architecture

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08.* Ieee, 2008, pp. 1–10.

[3] Y. Okamoto, T. Oishi, and K. Ikeuchi, "Image-based network rendering of large meshes for cloud computing," *International Journal of Computer Vision*, pp. 1–11, 2011.

[4] Y. Nimmagadda, K. Kumar, Y. Lu, and C. Lee, "Real-time moving object recognition and tracking using computation offloading," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on.* IEEE, pp. 2449–2455.

[5] Y. Cui, Z. Zeng, and L. Liu, "An object detection algorithm based on the cloud model," in *2009 International Joint Conference on Computational Sciences and Optimization.* IEEE, 2009, pp. 910–912.

[6] University of Auckland Imaging VCC [online] (accessed 17 Oct, 2011) `http://imaging-frontend.cs.auckland.ac.nz:8080/ImagingJSF`.

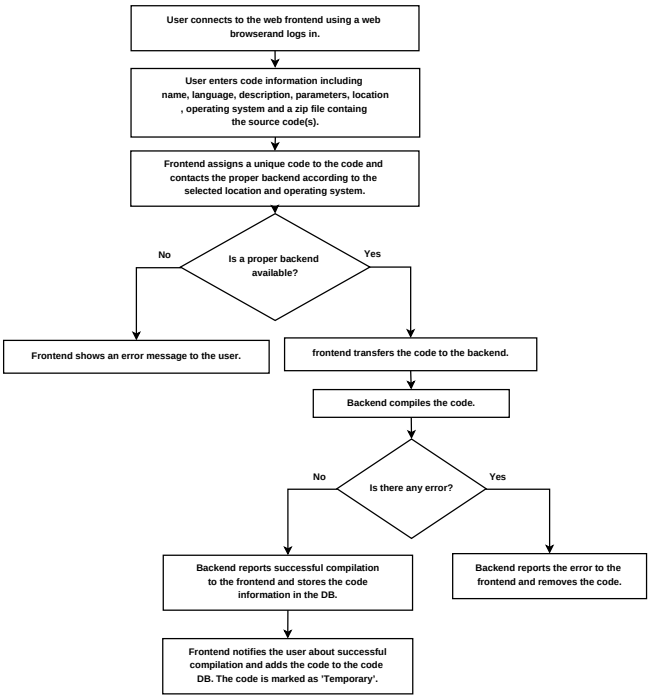

Figure 8. User interaction workflow: Uploading new source code.

Figure 9. User interaction workflow: Running a new experiment.