A simple drop and compute model for a SaaS cloud infrastructure for advanced research computing

The experiences of building OBOE, the Oxford Batch Operation Engine

Neil Caithness and Milo Thurston
Oxford e-Research Centre
University of Oxford
Oxford, United Kingdom
neil.caithness@oerc.ox.ac.uk; milo.thurston@oerc.ox.ac.uk

***Abstract*—We describe our experiences in building a web-based platform for cloud service integration for advanced community research computing within the context of a large funded project.**

Keywords-SaaS; AGILE; Drop & Compute; FP7

I. INTRODUCTION

The trend in advanced research computing is moving towards the delivery of community applications through a uniform interface and consistent user experience. The Oxford Batch Operation Engine [1] (OBOE) is a science gateway to research computing, with new methods available to developers to integrate new applications as software services, removing the need for them to build bespoke user interfaces or manage software delivery. For the user, OBOE provides easy and uniform access to well-known community research applications and new developments alike without the need for software installation or technical knowledge of running software in the cloud.

The principal aim of OBOE is effectively to decouple both the user and the application developer from the complicated software systems that make cloud and distributed computing possible, and yet to reconnect them again directly in their mutual research landscapes.

OBOE, or more specifically the web-based platform for cloud service integration, was built on a limited budget with the requirement that it be a lasting system operational and sustainable into the future. We believe that our experiences in building OBOE will prove useful to the broader cloud infrastructure community.

OBOE is the product of WP5 [2] of the ViBRANT [3] project funded by the EU/FP7; ~13 person months per year for three years running from December 2010 to November 2013, with no additional systems or compute funding. The ViBRANT project as whole is a large coordinated activity spread across 17 partner organizations in 12 countries.

The Objective for WP5 was simple: *To provide seamless integration of relevant external computing services for biodiversity researchers and Scratchpad* [4, 5] *users.*

The final Deliverable, against which success will be measured, is equally simple: *Refined and sustainable software services available for public use with mechanisms to measure usage rates.*

Between these two statements lie the details of paths chosen or rejected, judgment calls and pragmatic decisions, of instant-wins versus long-term usefulness and sustainability. We use these terms deliberately to emphasize what we believe is the central tension, or central experience: that the social context (in all its dimensions) overwhelms technical choices and decisions, and that, precisely for this reason, those technical choices and the scope of their influence need to be both protected and limited. In short, in our experience, simplicity and decoupling out-weigh almost everything.

II. AGILE

The ViBRANT project was, as far as we are aware, the first large project funded by the EU with a stated mandate to be AGILE [6]. (We believe BioVel [7], our sister project, was the second.) Note that the greater ViBRANT project was not entirely about computing, but about a research community (taxonomy and biodiversity) for whom computing plays a significant part.

The aspect of AGILE that appealed to project proposers and funders alike is that it encourages a flexible response to change (in the broader research landscape relevant to the community). The aspect that appeals to us, as systems and service designers and developers, is that requirements and solutions evolve through collaboration between self-organizing components. (Note that we choose the word ‘components’ deliberately as we mean something more inclusive and granular than ‘teams’, which might imply that we refer only to people.)

There is a consequence to adopting an AGILE mandate at the high levels of a large project: it creates a selection process for fine-grained, hierarchical, modularity. Certainly the benefits of modularity in software development, or even more generally in any collaborative activity, were recognized long before the AGILE manifesto, and modularity itself is not actually mentioned. There has also been much discussion on the failures of the AGILE program and questioning of its suitability to all kinds of collaborative activities, especially in large, complex and distributed projects [8]. But the manifesto does mention simplicity (#10) and self-organizing teams (#11). The lessons of our experience are that even the *expectation* of adaptability favors modularity, and that

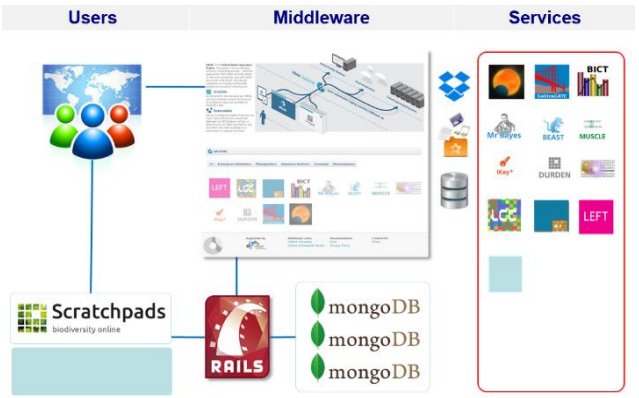

Figure 1. OBOE System Architecture

modularity simplifies and protects the technical decisions made within self-organizing components.

III. De-coupling

Our experience with OBOE shows that the mandate to be adaptable confers an advantage on modularity, and introduces certain freedoms that in an alternative world of functional specification might be considered extra constraints.

Fig. 1 above shows a traditional architecture diagram with three columns: *Users*, *Middleware*, and *Services*. At one level the diagram serves us well: it shows the relationship between modular, collaborating elements. The replicated array of mongoDB data-bases could be swapped for some alternative (given a compelling enough reason to do so [e.g. 9]) with little consequence elsewhere. In the empty box below Scratchpads, a new consumer of services could be "plugged-in", using the API published by Rails [10], and present an interface to a new community of users previously not engaged. (The BioVel project, mentioned before, might wish to implement this for instance.) All the elements in the left two columns are fairly traditional in their mode of collaboration (REST APIs in this case).

Where this traditional diagram serves us less well, is in making it difficult to emphasize the extreme decoupling we have chosen for the services. Each service in the palette is an independent and self-contained entity—independent in terms of machine and location (any computer, anywhere in the world) and independent in terms of operating system, software platform, and the like. There is no recognizable API protocol for interaction either, not REST, SOAP, WSDL, or any such contrivance that has proved so valuable in other areas (although we did at first experiment with several of these). In Fig. 1, in line with the divide between *Middleware* and *Services*, we show several icons representing shared storage: these act as conduits in our architecture. We call the mechanism Drop & Compute, discussed below in the next section.

We believe that a key element for the success of OBOE has been the alignment of our technical decoupling with a social decoupling as well. Although the authors share office space, swap experience, and collaborate in all the good senses of the word, we have divided development on OBOE strictly in line with the service decoupling and in line with our respective expertise: M.T. has designed, implemented and maintains everything in the Middleware column, N.C. has designed and developed most of the services from beginning to end (*most* of the services because, entirely as expected, some have been contributed by other authors. See the Services section below.) One of us prefers *nix based platforms and open source software systems; one of us prefers Windows and Matlab. These technical choices are protected by the scope of our decoupled modularity and make absolutely no difference to our collaboration.

IV. Drop & Compute

It is useful to digress briefly into a short history of Drop & Compute. At about the time we started making key decisions for the design of OBOE, it turns out that the team supporting High Throughput Computing at The University of Manchester via their very large (1600+ core) Condor pool were looking for a simplified way for users to access the resource. Their first solution, invented by Ian Cottam [11, 12], was to use shared folders on Dropbox: the user prepares a `.zip` file and drops it in the appropriate folder, the resource monitors the folder via a bash script and notices the new arrival, starts the processing, and returns the result for the user to pick up. In this formulation there is one Dropbox folder per user in a single Dropbox account, and there is very little middleware overhead, essentially just one bash script. Let's call this the Manchester style Drop & Compute where the shared storage is exposed to users.

Manchester HTC soon bumped into the limitations of a single Dropbox account with many folders and changed over to each user having their own Dropbox account. Some users even chose to purchase their own 50GB or 100GB accounts.

The next, and more severe problem was not technical, but a social concern over security: users were concerned that by using this mechanism they lost control of the confidentiality of their research data. Dropbox uses commercial, USA-based servers, and the US Patriot Act means that US companies must surrender any data they hold if requested to do so by Federal Government agencies. In response Manchester implemented a 'local version' of Drop & Compute, where the user merely has to mount a folder on the submit node on their local computer, and continue with the same drag-&-drop approach to submit jobs. Ian Cottam notes the loss of several desirable features, mostly for the user experience, in adopting a local disk mount [11].

Following Manchester's lead, other large system operators also began investigating Drop & Compute for user job submission [e.g. 13].

At Oxford we were looking for a straightforward and convenient way for a service to receive job parameters from the middleware (not directly from the user). We had already put in place a conventional system of wrappers for proprietary programs, but this seemed too tightly-coupled and we were already responding to the AGILE expectation of change. We too started using Dropbox. In our version, it's the OBOE application that packs content gathered from web-forms, user supplied files, and any other relevant information into a `.zip` file and drops this into the appropriate folder, not the user. The

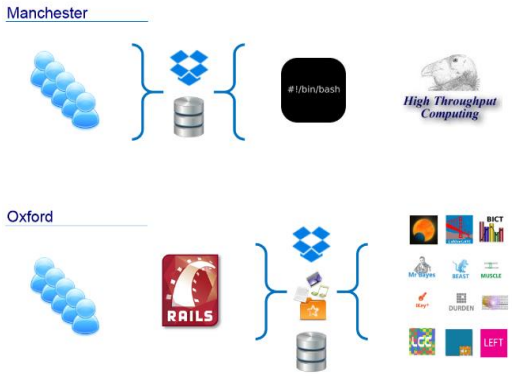

Figure 2. Manchester versus Oxford Styles

shared storage is not exposed to the user. For us there is one Dropbox folder per service, not per user, and instead of a single bash script there is an entire web-application complete with APIs and replicated data-bases managing user accounts and service requirements. Let's call this the Oxford style Drop & Compute. Fig 2. above illustrates the different ordering between the Manchester and Oxford styles.

Just as at Manchester, and also in response to users' concerns about confidentiality, we soon changed as well, or rather we added an alternative to Dropbox: a hosted private instance of SparkleShare. Users are unaffected, but services need to accommodate the change to the shared storage by installing a local node of SparkleShare.

Finally, for convenience, and because many of the OBOE services run on virtual machines hosted in the OeRC, we now also use a shared GPFS which our various systems can mount as either NFS or CIFS. This is a rather acronymic way of saying that we have a convenient and fast way for our local systems to share files, and an equally convenient but perhaps not-so-fast way for services on remote machines to share files. Users are unaffected by any of this, and choices are made on a per-service basis.

V. The OBOE web-application

OBOE is a Ruby on Rails application using MongoDB as its data store. Ruby on Rails is regarded as a framework allowing the rapid and agile development of web applications and was chosen for this reason once the decision was made to use a separate framework rather than building Drupal plugins for use with Scratchpads. In this section we explain our decision to develop a separate framework.

1) A separate application allows us to give access to a wider audience of users than those just using Scratchpads.

2) A major function of OBOE is to provide a REST interface for applications that do not have one; writing Scratchpad plugins in Drupal would not have achieved this.

3) Writing a separate application with decoupled modes of interaction protects the technical choices made and limits their scope.

The OBOE rails application is simple, containing only one class, the "job". Each job represents a single analysis requested by a user, no matter what the service or the input data. Each job type has an associated module which defines two methods; one starts the job running, the other checks whether it has completed and obtains the final data. Each time a job is started a progress check is queued (the interval varies by job type); if a job is found to be running then another check is queued, and so on. Successful completion or reported failure will complete processing of the job and no further checks will be made.

Details of the required data are defined in the job model and may be queried via the API. The web interface presents a form (generated from the same definitions presented by the API) which makes clear the required information for that type of job.

MongoDB was selected as the database as each job is essentially an independent analysis which is either newly created, running or finished, and there is no need for complex queries on jobs stored in OBOE's database. There has been recent criticism of MongoDB [e.g. 9] as being a poor replacement for relational databases, but OBOE is a case where a relational database is not required. MongoDB's flexibility for the storing of arbitrary data is ideal for the variety of jobs which it makes available, where the main requirement is to pass the user's data from the user to the application and back as efficiently as possible.

VI. The OBOE services

OBOE services fall squarely in the category of *Software as a Service* (SaaS) where software and associated data are centrally hosted in the cloud. Our intention has always been to create a platform for the authors of research computing applications to make their work more easily accessible to users without incurring any of the overheads and disadvantages of traditional software distribution, or requiring knowledge of cloud computing. From the user's perspective, access to the latest research computing applications is made available simply via a web-browser. Neither of the end-to-end parties—the application user or the application author—need care about any of the intermediate cloud technicalities.

Technically, OBOE creates a REST interface for any self-contained batch processing application which can then be used more widely.

VII. Usage Rates

Usage rates of OBOE (see Fig. 3 below) have grown remarkably consistently over the three year duration of the project to build the system. We have not been overwhelmed by scaling issues, nor do we find we are underexposed.

VIII. Discussion and Conclusions

In all of this discussion, we need to make it clear to users and application developers alike that we are talking essentially about non-interactive applications. Perhaps this is the single most important point of misunderstanding of the OBOE model: that perhaps *batch* does not meet users' long-term expectations of research computing applications.

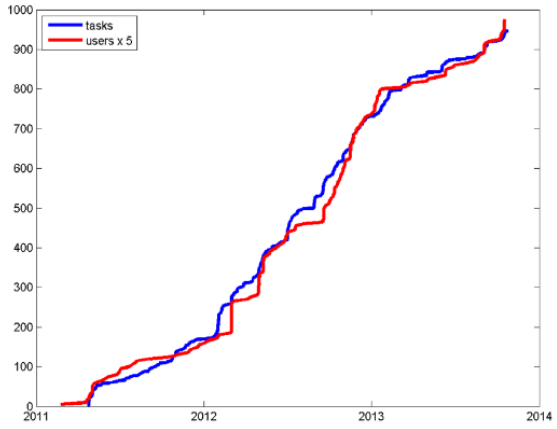

Figure 3. OBOE usage rates

Of course interactive applications are hugely important and it would be misguided to argue otherwise, though batch computing will not fall by the wayside just yet. Over the next decade, as efforts to model the biosphere are taken up seriously [14, 15, 16], we believe that parameter sweeps for an astonishingly diverse array of biological and environmental models will become commonplace. To meet this growing computational requirement we believe that batch computing, and something very like the OBOE model of service integration, will play an increasingly important role.

ACKNOWLEDGMENT

The authors received funding from the ViBRANT project, European Union 7th Framework Programme, Research Infrastructures group, Contract no. RI-261532. Compute, storage and systems support were provided, without additional external funding, by the Oxford e-Research Centre, University of Oxford. We thank colleagues in the OeRC and the ViBRANT project for helpful discussions.

REFERENCES

[1] N. Caithness and M. Thurston, "The Oxford Batch Operation Engine," https://oboe.oerc.ox.ac.uk/, 2013.

[2] N. Caithness, et al. "WP5: Data Interaction and Services," http://vbrant.eu/content/wp5-data-interaction-and-services, 2013.

[3] V. S. Smith, et al. "Virtual Biodiversity Research and Access Network for Taxonomy (ViBRANT) ," http://vbrant.eu/, 2013.

[4] V. S. Smith, et al. "Scratchpads 2.0: a virtual research environment infrastructure for biodiversity data," http://scratchpads.eu, 2013.

[5] V. S. Smith, et al. "Scratchpads 2.0: a Virtual Research Environment supporting scholarly collaboration, communication and data publication in biodiversity science," *Zookeys*, 150, 2011. Special Issue: e-Infrastructures for data publishing in biodiversity science 53-70, doi: 10.3897/zookeys.150.2193.

[6] K. Beck, et al. "Manifesto for Agile Software Development." Agile Alliance, http://agilemanifesto.org/, 2001.

[7] A. Hardisty, et al. "Biodiversity Virtual e-Laboratory (BioVel)," http://www.biovel.eu/, 2013.

[8] P. Kruchten, "Agile's Teenage Crisis?" http://www.infoq.com/articles/agile-teenage-crisis, 2011.

[9] S. Mei, "Why You Should Never Use MongoDB," http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/, 2013.

[10] M. Thurston and N. Caithness, "The Oxford Batch Operation Engine API," https://oboe.oerc.ox.ac.uk/docs, 2013.

[11] I. Cottam, "The Evolution of 'DropAndCompute'," http://www.walkingrandomly.com/?p=3339, 2011.

[12] D. DeRoure, "Drop and Compute," http://blog.openwetware.org/deroure/?p=97, 2011.

[13] J. Lander, "From desktop to grid," http://nationalgridservice.blogspot.co.uk/2010/04/from-desktop-to-grid.html, 2010.

[14] A. Hardisty, et al. "A decadal view of biodiversity informatics: challenges and priorities," *BMC Ecology*, 13:16, 2013.

[15] A. Hardisty, "Horizon 2020: A call to forge biodiversity links," *Nature*, 502:171, 2013.

[16] D. Purves, et al. "Ecosystems: Time to model all life on Earth," *Nature*, 493:295-297, 2013.