Architectural and Technological Aspects of the Cloud Data Analysis System Development, Case of ISTINA System

Valery Aleksandrovich Vasenin

Doctor of Science, professor, head of department, Lomonosov Moscow State University
+7 (495) 939-21-59, vasenin@msu.ru

Maxim Anatolyevich Zanchurin

Junior researcher, Lomonosov Moscow State University
maxim.zanchurin@gmail.com

Alexander Sergeevich Kozitsyn

Ph.D., leading researcher, Lomonosov Moscow State University
alexanderkz@gmail.com

Maxim Aleksandrovich Krivchikov

Ph.D., senior researcher, Lomonosov Moscow State Universivy
maxim.krivchikov@gmail.com

Dmitry Alekseevich Shachnev

Postgraduate student, Lomonosov Moscow State University
mitya57@gmail.com

1 Introduction

In the present paper, the architecture and software development process of informational-analytical system ISTINA (Intellectual System of Thematic Analysis of Scientometrics Information [1]) are discussed. ISTINA is used in production in Lomonosov Moscow State University and, as a pilot project, by a number of universities and research facilities of Russian Academy of Sciences (RAS).

In its present version, ISTINA represents the integrated cloud data processing and analysis decision support system in the field of scientometrics. Data integration mechanisms of the system are implemented in terms of a relational database which represents the primary components of the ontology and the interacting applications. Presently, the Oracle database is used. The system is accessed by means of web-interface. The interface is built with Django framework [2]. It is running on the uWSGI application server [3] and Nginx web server [4]. The source code of the system is written mostly in Python, JavaScript and C++ programming languages. The

system is designed in accordance with the methodology of Django framework. Parts of the system concerned with the processing of closely related domain concepts are written as the separate “applications”. For example, the concepts of a dissertation, its affiliation with the author, her scientific advisers and reviewers are integrated into the “dissertations” application. The subsystem in terms of the system design consists of several interconnected applications. For example, “Dissertation Councils” subsystem consists of “dissertations” and “dissertation_councils” applications. Application code is structured in accordance with Model-View-Controller (MVC) architecture, with an exception in terminology. The “view” in terms of Django framework is responsible for the processing of user requests, which corresponds to the “controller” in terms of MVC architecture. The “view” in terms of MVC, correspondingly, is contained inside Django “templates”. Integration with the external systems (for example, Web of Science and Scopus bibliographic databases) is implemented as external services which either use the database directly or by means of domain abstractions of the system core code. For asynchronous continuous data processing, we use Celery task queue. The application server is using Memcached [5] server for the cache. Recently we faced an issue with the maximal object size restriction in Memcached. It is possible that this component will be replaced in the long term.

2 Source code management

Since 2017 we use GitLab system [6], which provides the web interface for source code repository management for ISTINA. Our GitLab instance is hosted in Lomonosov Moscow State University data center. One significant feature of GitLab which is used extensively in our software development process is code review. In the same way as with proprietary GitHub system, GitLab supports so-called “merge requests” – series of source code changesets together with the justification of changes. Each merge request is developed in a separate branch of version control system, independently of the other system code changes. On completion of the development of some new feature, the developer assigns a reviewer to another team member who was not involved in the development of this feature but has the knowledge of the affected parts of the code. Merging of changes in the production branch of the code without the review is forbidden. Such a process has the following advantages: all team members are able to view the recent code changes grouped by the domain task; the reviewer may spot typos and at least the trivial errors in code.

For the source code modification team members use a diverse set of text editors and integrated development environments, including the community edition of JetBrains PyCharm and also Vim, Nano and Mcedit text editors in Unix environment. Some members of the team previously used Eclipse PyDev have switched to PyCharm due to more simple and reliable configuration process in PyCharm. Other useful tools which are worth mentioning are IPython interactive command-line and Jupyter Notebook [7] application. These tools are used for rapid prototyping and manual testing of the code and in complex data analysis problems.

In the context of development process modernization, we are planning to introduce the Continuous Integration practices. For each merge request, GitLab automated test framework will run the set of tests, including the basic smoke test which ensures that the core components of the system are working properly and that it is possible to load the main page of the web interface. The smoke test is intended to detect the errors in system configuration, for example, in case of new software library is used without its inclusion in the dependency list of the system. The second part of tests will include the static analysis of the code with PyFlakes tool. This tool detects typos, syntax errors, variable naming and external module import errors. The last part of tests will include unit-tests and integration tests from ISTINA test suite.

Future plans in software development process include the hybrid distributed polygon for automated testing of the distributed system in the whole and the test coverage tools which ensure that the test suite covers all important parts of the code.

3 User support

The ISTINA system is used by the largest scientific and educational organization in Russia, Lomonosov Moscow State University. Currently, 15 organizations and more than 25000 users are working in the system. Given the small size of the development team, the traditional ways of communicating with users (such as phone calls, email messages, and personal communication) have proven to be ineffective. Because of this, it was decided to deploy a system for processing users' requests based on Redmine [8]. Every page of the ISTINA system interface has a "Create a request to the users' support service" link at the bottom. When user clicks that link, a dialog window is opened, where one can briefly describe the problem and select the subject category of the request. When the "Submit" button is clicked, the ISTINA system server creates a new ticket in Redmine using the application programming interface (API), which contains the text entered by the user and some additional information. This information includes the name and the place of employment of the user and also the address of the system interface page from which this request was submitted. The Redmine is configured to send email message to the user confirming that his request has been accepted. By replying to such email the user can provide any additional information or attach a file to the ticket.

The Redmine system is also used for planning new tasks for modernizing and supporting the system. Work is in progress to allow the users responsible for maintaining information in the ISTINA system ("representatives") in the client organizations to access the Redmine system. This way representatives will be able to do the primary filtering of requests and reply to some non-technical questions, for example to administrative or organizational ones, and to the questions that they can solve themselves.

For managing the runtime errors, ISTINA uses the Sentry continuous monitoring system. When an error occurs in the system (in terms of Python language, an exception is raised), a new issue is created in that system, and the debugging information is attached to it, such as the HTTP request parameters, the traceback and the values of variables. Issues which have the same traceback are grouped together, which simpli-

fies their analysis and prioritization. In practice, this continuous monitoring module allows the developers to react quickly when regressions in the code occur.

4 Transition from Django 1.4 to Django 1.8

The common practice for large modern software systems is using the frameworks created by third-party developers. Usually, such frameworks are open source projects which are supported by one or several large companies. Examples of such open source frameworks for different programming languages are ASP.NET Core MVC for C#, Spring Framework for Java, Rails for Ruby and finally the Django framework for Python which is used in ISTINA system.

The application programming interface (API) of the used framework is to a certain extent integrated with a large part of the system source code. Because the framework itself is evolving rapidly, some features get removed or change their behavior with every new version of the framework. Support of the previous version, which means fixing bugs and sometimes security vulnerabilities, usually terminates in a few years, except for some releases which are named long-term support releases (LTS). If the system uses a deprecated version of the framework, the modification of the system source code is required. For dynamically-typed programming languages, the modification is complicated by the absence of feedback from the compiler which allows direct detection of the code fragments which need modification.

During spring of 2017, we have transitioned from old and unsupported Django version 1.4.x to the long-term support version 1.8.18. Because the ISTINA system is rapidly developed, there was a requirement to maintain source code compatibility with Django 1.4 to avoid the divergence of the source code state between production and development branches. The transition was done by gradually adding support for 1.5, 1.6, 1.7 and 1.8 versions, and dropping support for 1.7 and earlier versions at the end of the transition.

When most of the changes for Django 1.8 support were ready, there was a three-week beta testing period, when most of the developers started using Django 1.8 locally, and a demo server was upgraded to 1.8. A simple script was written to query the most used URLs on the test server. During this stage, many bugs were detected and fixed. The final stage was adding a new virtual environment on the production server and switching requests to it. During this stage, some more bugs were found and fixed in a few days.

Django has a two-cycle period of deprecating and removing parts of the API: if a feature is marked as deprecated in some release, then it is removed not earlier than in two major releases after that. For example, it can be marked as deprecated in 1.5 and removed in version 1.7.

The changes needed to ensure compatibility with new Django can be divided into the following groups:

Updating dependencies. For those dependencies where the upstream versions were used unchanged (“external dependencies”, for example, the django-select2 library which provides an HTML widget for selecting objects, or django-reversion library

which provides support for tracking changes to objects in the database), they were upgraded newer upstream versions where possible. For those dependencies where the upstream sources were changed and for dependencies written from scratch for the ISTINA system ("internal dependencies", for example: the userprofile library which provides structures and templates for users' profiles, and sqlreports library which formats the results of templated SQL queries for user interface), support for Django 1.8 was added and the dependency version in ISTINA system requirements was updated. Some dependencies have been removed because they are no longer required with Django 1.8. One new dependency was added because it was split out of Django code (formtools).

Getting rid of deprecated modules and APIs. Some modules could be replaced with their modern equivalents without losing Django 1.4 support, such as the switch from django.utils.simplejson module to json module from Python standard library. This also includes all features which were deprecated in Django 1.4 or earlier. In other cases it was possible to use different code depending on Django versions, using either 'try ... except' or 'if ... else' constructions, for example using transaction.commit_on_success function decorator for atomic database transactions support for Django 1.5 and older versions, and transaction.atomic for Django 1.6 and newer versions.

Rewriting the code. where there is no equivalent to the old APIs. For example, Django 1.5 removed some old function-based views (such as direct_to_template or object_detail). All code that was relying on these views has been ported to class-based views (such as TemplateView and ListView).

Finally, we had to change the code of Django framework itself in two places, fixing support for Oracle database backend. One of the patches was coming from then newer version upstream, another one was forwarded to Django developers and later applied there.

Most of the bugs during the final stages of transition were detected using the Sentry system. These bugs were generally easy to fix because they have stack traces available to look at. Some other bugs were much trickier to detect: for example, it was found out that ordering of fields in forms has been broken, and some unwanted fields showed up to users in some cases. This kind of issues could only be detected by comparing the same web page opened with Django 1.4 and Django 1.8.

Upgrading the framework and other third-party dependencies is not a one-time action. For example, the currently used Django 1.8 version will be supported only until April 2018. It is worth noting that updating the code for a newer Django version is a "technical debt" of the system. The future plan includes getting rid of all features deprecated in Django 1.7 and 1.8, updating all external dependencies to their latest upstream versions, and then gradually migrating to Django 1.11 long-term support release before April 2018. It is also worth noting that since version 2.0 (which is following version 1.11, the release is planned to December 2017) Django developers have switched to the new development cycle which simplifies the transition for projects which use the long-term support Django versions.

5 Using Sphinx for building the project documentation

One of the aspects which are often not given enough attention when developing large and rapidly evolving software systems is creating and maintaining the system documentation. Previously the documentation for ISTINA system was developed with LaTeX tools (reports for research projects) or editors compatible with Microsoft Word (documents conforming to the Russian standard for program documentation, ESPD). The fact that Microsoft Word documents are stored in the repository opaquely, without the ability to view modifications, led to the documentation and the actual code being not synchronized with each other. In the beginning of 2017, the process of preparing the documentation for ISTINA in ESPD format (manuals for users, programmers, and system administrators) was switched to using the Sphinx documentation generator [9].

Sphinx is a tool which uses a lightweight reStructuredText markup language as its input. It can build the set of documentation with export to different formats. Sphinx is used by many software projects, including the Python language, Django framework, and Linux operating system kernel.

For the user documentation the main advantage of Sphinx is its ability to generate outputs in multiple formats, most importantly PDF and static HTML pages with support for searching by the means of client-side JavaScript code. The HTML version of the documentation is published on http://docs.istina.msu.ru/ website and is updated from the same sources as the production code. The PDF version was used for preparing the ESPD documents and was printed in a limited number of copies.

For generating the code documentation, the built-in facilities in Sphinx for documenting the Python code were used. Sphinx allows one to build documentation based on documentation strings (docstrings) in the Python code, which means that documentation can be written at the same time when the code is written. The format for docstrings is described in a Python enhancement proposal, PEP 287, and thus is familiar to all Python developers. There are various directives like automodule, autoclass, autofunction, etc. which insert the documentation for Python code into the reStructuredText document.

The ISTINA developers have implemented an extension for Sphinx which adds support for documenting Django models. There is no need to write docstrings at all, all the needed information is extracted from the models' metadata.

The ISTINA developers have also implemented a Django command to import our code and build the Sphinx documentation. It can be called as manage.py build_sphinx [-b builder].

Both our Sphinx projects are using the ReadTheDocs HTML theme (sphinx-rtd-theme) and localized into Russian language. We have contributed code to upstream Sphinx to add support for Russian typographic quotes and to fix the index generation for words with Cyrillic letters.

6 Conclusion

The design, development and maintenance processes for cloud (software-as-a-service) software systems have some distinctive features which distinguish it from the traditional software product lifecycle. These features include the rapid continuous development process, the absence of releases, the need for large-scale user support system. The aforementioned architectural and technical solutions and tools for developing, maintaining and improving the ISTINA system have shown their efficiency and promise good perspectives for the future.

References

1. Sadovnichiy V.A., Vasenin V.A., et.al. Intellektualnaya sistema tematicheskogo issledovaniya nauchno-tekhnicheskoy informatsii («ISTINA»). Moscow, 2014. 262 p (in Russian).
2. Django: The web framework for perfectionists with deadlines. Online. URL: https://www.djangoproject.com/, accessed 01.06.2017.
3. The uWSGI project — uWSGI 2.0 documentation. Online. URL: https://uwsgi-docs.readthedocs.io/en/latest/, accessed 01.06.2017.
4. Nginx. Online. URL: https://nginx.ru/en/, accessed 01.06.2017.
5. Memcached — a distributed memory object caching system. Online. URL: http://memcached.org/, accessed 01.06.2017.
6. GitLab: Code, test and deploy together with GitLab open source git repo management software. Online. URL: https://about.gitlab.com/, accessed 01.06.2017.
7. Project Jupyter. Online. URL: http://jupyter.org/, accessed 01.06.2017.
8. Redmine: Overview. Online. URL: http://www.redmine.org/, accessed 01.06.2017.
9. Sphinx 1.6.3+ documentation: Overview. Online. URL: http://www.sphinx-doc.org/en/stable/, accessed 01.06.2017.