INFORMATICA, 2014, Vol. 25, No. 3, 467–484

DOI: http://dx.doi.org/10.15388/Informatica.2014.24

Advantages of Deferred Approach for Time-Critical Tasks

Pavel OSIPOV*, Arkady BORISOV
Riga Technical University, Information Technologies Institute
Meža st. 1/4, LV-1048, Latvija, Riga
e-mail: pavels.osipovs@gmail.com, arkadijs.borisovs@cs.rtu.lv

Received: September 2012; accepted: November 2013

Abstract. This article discusses about ways of organizing and using the system for typical user behavior model construction within a distributed information system using Python servers. Much attention is paid to analysis of the issue of effective organization of the research process using the Python language in all its phases.

Possibilities of using Python servers implement different policies for problems with strict limitations on the processing of incoming requests was reviewed. Along with the classical approach, we consider a new variant of the organization of the server: Deferred, which allowed to stay within the constraints of the target problem domain.

Key words: behavior model, strict resource limits, deferred, Python, Python server policy.

1. Introduction

Currently, there are a lot of data processing algorithms that require large hardware resources for the operation. This is due both to an increase in complexity of the algorithms and the volume and structure of the processed information. The usual practice in such cases is to increase the number and capacity of machines that produce processing, in particular, move calculations to a clusters or cloud computing. But not always such an approach is justified economically, and the question arises about the performance of such tasks on the available equipment, but using different optimization techniques. In particular, the research capabilities reduce the complexity of the algorithms can yield much better results than the use of more efficient equipment.

The purpose of this paper is to evaluate the effectiveness of the algorithm for constructing a model of user behavior, as described in Python, as well as the impact of various policy servers that implement this algorithm, the number of resources required for the Detection of Abnormal Activity (AAD).

In the first part of the article we discuss about ways of unification of researchers tools to move from a heterogeneous multi-component research environment to the uniform. Such a transition is required to increase the productivity of research activities as well as

*Corresponding author.

minimize the costs of establishing and supporting the intermediate layers between the different technologies used. As a consequence, also decreased the number of errors that occur when converting data between different modules and formats.

The second part describes the structure of the developed module, used algorithm, performance requirements and the target Information System (IS) restrictions.

The third part discusses about possible approaches of the implementation of such modules in a distributed IS.

The fourth part is justified used approach to create structure of the AAD module, the features of his implementation, and used technologies.

The fifth part describes testing the key modules of the target system for compliance with the original requirements and restrictions.

In conclusion, the possible ways to increase the efficiency of the module discussed. We describe the potential for increased efficiency through change and configuration used in the implementation of third-party Python modules.

2. The Effectiveness of the Research Environment

In the process of modern scientific research typically used a large number of tools for generating, processing, analysis and visualization of data. Often used for each specific sub-tasks, used software what focused only on a narrow range of problems of this type. However, with increasing number of such programs is becoming increasingly difficult to transfer data between a variety of analysis tools, each of which can use his personal format for input and output data. An important feature is the tracking data transformations made in the process of research, to have a clear picture of how the final results of the experiment depends on the source. It is also important to track the stages on which may produce error, to understand its causes: human error, the error in the input data or the analysis module.

Different tools are realized in different programming languages: C, C++, Fortran, Cobol, Perl, or as a proprietary (closed) software packages such as MatLab, MatCad, Mathematica, Statistica, SPSS. Such a large number of different modules in the end greatly complicates the whole process of research. A further important disadvantage is the difficulty or the impossibility of establishing direct communication between modules in order to exchange data (pipelines) without the use of intermediate results files. As a result, a significant part of the researcher resources can be devoted to solving the problem of converting data from one format to another.

All this not only complicates the process of research, but also increases its duration, and increases the likelihood of errors.

It is clear to assume that the using a single common environment in which the experimenter can implement most of their needs will achieve better results, as research resources will be spent on synchronization of data between different technologies.

One of the possible ways to solve this problem is to use programming language Python (Lucks, 2008; Millman and Aivazis, 2011), as well-established and highly-loaded in the creation of systems and to implement a variety of scientific problems.

Python was used as a common research platform. Interesting, that at the present time, there were recommendations to use Python code instead of the classical pseudo-code, since its syntax is often more expressive when describing complex algorithms, which gives a high level of self-descriptions of documentation. Currently Python present opportunities for the implementation of any phase of the project activities within a unified approach and standardized data types and formats. There are modules to generate, analyze, modify, and visualize data, and more importantly, himself Python, being a general purpose language, is used to determine the relationship between these modules, and to manage and analyze data streams.

There are also methods to increase the speed of programs in Python, such as the implementation of the performance-critical areas in the form of modules written in C (Bradshaw *et al*., 2013) or use of Just-in-time (JIT) compilation of Python code (Chang *et al*., 2011).

In turn, using the Python language can cause some specific problems. In view of the fact that what by its nature, this language is interpreted with dynamical typing, there is a threat of run-time errors. Reduce their influence can be done by covering most of the code by automated tests.

Based on analysis above Python language was selected to implement target system of this research.

3. The Structure of the Developed Module

We proposed (Osipov and Borisov, 2011) the implementation and evaluation of the algorithm design and analysis of user behavior in the electronic form of Markov model based on a graph, in order to detect its abnormal activity. It is shown that the algorithm used can be applied in systems that operate on sensitive data. However, a test system, implemented in the programming language PHP, showed an extremely low computational efficiency of processing models. It was also very difficult to implement, thereby reducing its flexibility and made it difficult to make changes that are often required during the experiments.

The design of the target system described in this paper lies in the fact that at one time many users perform multiple actions (transactions), each of which, before being processed by the main system, must be approved by the security module. During the processing of each transaction security module, in particular, examines how different the above request from the typical behavior for this user. For this transaction is calculated metric (Osipov and Borisov, 2011), to calculate it, user must have a personal model of behavior. In the case of the first login it is assigned is common for this user type model. While working, based on some rules, model changes adjusting to the peculiarities of the specific of individual behavior of the target user.

The model of user behavior is presented as a Markov Chain (MC), which stores information about the frequency of accesses to different data sources and preferences as of user same type group as and the personal behavior of each individual user (Albert and Barabási, 2002).

Establishment of a typical model takes time and can be produced separately from the main process of the system functioning. The most critical time is just an analysis of current behavior. This operation is performed *for each* user transaction.

4. The General Algorithm

Initially introduced variables X and Y having both values of 0. *Current state* called vector of the identifiers (*trace* α) recent transactions requested by user having fixed length. Each next step changing the $\langle$*current state of*$\rangle$ by adding code of done transaction to the end of the current transaction vector and removing one from beginning.

Based on previously developed MC for each step, calculated metric $\mu(\alpha)$, which determines the status of the current state, and the variables X and Y. At each step there are two options for calculating the value of the metric.

In the model there is an arc of the graph of the transition from the previous state in the current: $\beta_i \to \beta_{i+1}$. In this case, X and Y are updated following functions-parameters:

$$Y = Y + F\big(s, \big(s, s'\big)\big),$$
$$X = X + G\big(s, \big(s, s'\big)\big).$$

In the model there is a no arc what describes transition from the previous state to the current. In this case, X and Y are updated following rules:

$$Y = Y + Z,$$
$$X = X + 1,$$

where Z – is constant.

It then computes the value of the metric $\mu(\alpha)$, which is equal to Y/X.

The metric $\mu(\alpha)$ shows how well the MC predicts a *trace* α, that is, the smaller its value, the more accurate MC predicts a trace. Since μ is parameterized by the functions F, G and the constant Z, then a different choice of F and G will change the value of the classifier, what gives the ability to fine-tuning it to the specifics of the current domain.

The classifier can be constructed from the metric μ as follows:

$$f(a) = \left\{ \frac{1, \quad \mu(a) > r}{0, \quad \text{otherwise}} \right\}.$$

That is a *trace* α is classified as abnormal if the metric μ for the current transaction exceeds the target threshold value of r.

4.1.1. *Options for Computing Functions: The Parameters X and Y*
There are several methods for calculating these functions (Osipov and Borisov, 2011), which are chosen depending on the characteristics of the subject area. It is also possible to calculate several ways and to identify the presence of abnormality in at least one of the ways. Currently used only one approach: **Probabilistic metric (probability of failure to reach the goal)**.

Formulae for calculating the F and G are as follows:

$$F\big(s, \big(s, s'\big)\big) = \sum_{s_1 \in succ(s) \wedge s' \neq s_1} P(s, s_1),$$

$$G\big(s,\big(s,s'\big)\big)=\sum_{s_1\in succ(s)}P(s,s_1)=1,$$

$succ(s)$ – is the set of all states from which there is a nonzero probability of being in state s. Since the Markov chain, the following conditions:

$$\sum_{s'\in succ(s)}P\big(s,s'\big)=1.$$

4.1.2. *Evaluation of the Base Algorithm Complexity*
Evaluation of the asymptotic time complexity of the target algorithm is based on the following main characteristics:

- The number of nodes of the graph is denoted l, the number of arcs m, the current length of the trace s.
- Access to the arc between two arbitrary nodes of the target graph. Since the graph is represented in memory as adjacency list, we obtain information on the availability of the arc does not require a complete traversal of all nodes of the graph and in the general case is $O(\log l)$.
- Getting all the nodes of a graph with an arc connected to the target node, this operation uses the worst case a complete traversal of all arcs of the graph, estimate the complexity is $O(m^2)$.
- Operations on the trace have a linear dependence on the length of the track $O(s)$.

We assume what $n=s\log(l)m$, when the final view of the complexity in worst case will be presented as $O(n\log n^2)$.

5. Requirements for the Processing Speed for Each Transaction

Problem domain have very strict limitations for the each transaction processing speed: the upper threshold for 100 models processing is 500 ms. There is also a requirement to use typical server hardware without purchasing costly high-technology. Depending on the computation power of hardware calculation speed of one model may vary. At present an operation for computing metric and update the model, implemented in Python, uses about 50 ms of CPU time on a computer Intel Core i3 560 + 4 Gb RAM DDR3 running on Ubuntu OS. It should be noted that despite the fact that physically the Intel Core i3 560 has only two cores, but with Hyper-Threading technology, each of the cores can simultaneously perform two workflow threads fully transparent to the operating system. However, the addition should take into account the time spent on initializing Python interpreter, temporary costs of loading the model from the database to obtain the parameters and the issuance of the result. In sum, the time may reach more than 200 ms on a script, which does not consistently treat for 500 ms, 90 models.

There is a need to optimize the processing, use of parallel computing, using all possible cores, creating pools of database connections, using effective methods of long-term storage of models, as well as the efficient allocation of resources to handle a large number of concurrent requests.

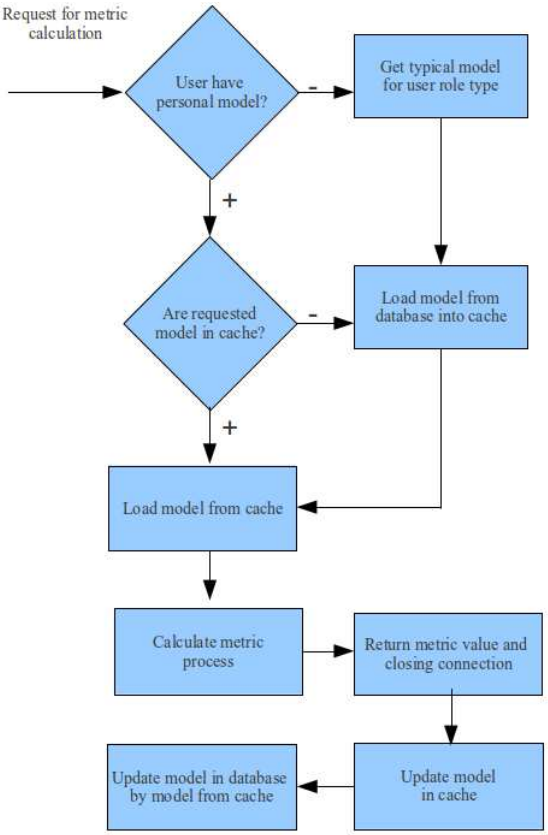

Fig. 1. The algorithm of module functioning.

Because of these limitations, it was decided to use the experience and some software tools used to create electronic systems targeted at heavy loads.

6. The Overall Structure of the AAD Module

The developed module is a self-contained part of a large information system security, which imposes certain restrictions on the possible architectural solutions for its implementation.

The target system is a set of distributed services, and security module is one of its components. In turn, the structure of the module construction of models of user behavior and evaluation of each action on the basis of this model (Fig. 1) is a sub module of the security system.

7. Review of Existing Approaches

Typically all servers what are implemented in Python, using one of two classical approaches to the logic of the work:

- use of **system process** (*heavy servers*) to handle incoming requests;
- use of **lightweight threads** (*lightweight servers*).

8. Creating a System Process to Handle Each Request

In the case of such an approach when a server receives request he creates a system process that handles the request regardless of the server itself. This allows to efficiently handle many simultaneous requests. However, the establishment of a system process is fairly resource-expensive operation, and often OS have limit on the number of simultaneously available to the application processes. Their number is rarely more than 100, even on powerful modern servers. This is mainly due to the relatively high costs for planning the implementation process by the operating system.

Alternatively, to save time in the initialization process, the server can use pre-created pool of system processes, what increases efficiency, but does not eliminate the problem of limiting the number of available processes.

9. Using a System Threads to Process Each Request

Depending on the implementation of the threads in the used OS the amount of resources to create a thread can be roughly equal resources to create a system process, or much smaller. However, in this case, threads count limit just bigger, and maintain a large number of simultaneous connections (from a thousand and above), this model may prove to be unworkable for the following reasons: consumption of address space on the stack for each thread, a large load on the scheduler and the limit on the total number of threads in the system.

It is seen that the second approach can handle the simultaneous number of connections within the given requirements. However, there are some drawbacks.

With increasing load requirements may change.

Physically, the server takes a lot of resources to support threads, but the main burden still lies on the modules to work with models and the need to consider the total load.

In view of these shortcomings, there is requirement of more optimal way to use server resources to handle a large number of simultaneous requests.

10. Deferred Approach

Recently, more and more popular becomes approach called *Deferred* (Samek, 2008). Its essence lies in the fact that when a request is called a module that is responsible for its processing, assigns a function – an event handler "*Calculation Completed*" and then the server "forgets" the received request and does not spend resources to support it. After some time, when the request is fully processed, the server receives a signal of an event,

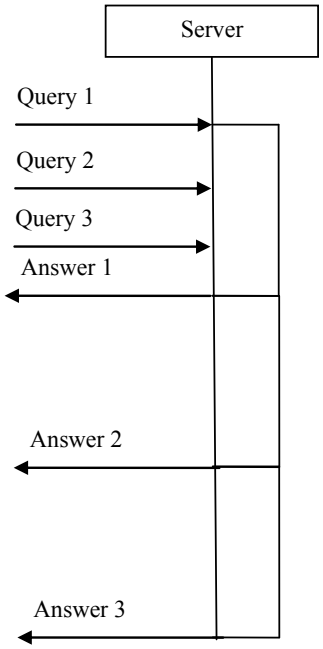

Fig. 2. Processing of requests the server.

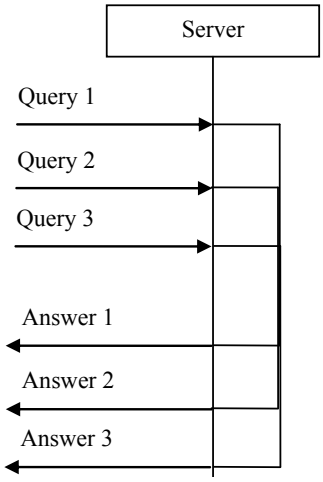

Fig. 3. Deferred by using remote services.

"*Calculation Completed*" and the calculation result, previously attached handler function is called and serves the answer.

Deferred concept differs from the typical methods also from the software implementation. In the code the server called special function, which uses data from remote services. Since the ideology itself does not imply *Deferred* overhead waiting for an answer, this function immediately returns the result, despite the fact that he had not yet calculated. This is achieved by returning the object of a special type of **deferred**, which adds the necessary function hooks (at least handlers for received result and received error (if any), and then almost all of the resources spent on processing the request, shall be cleared. The next step occurs only at the time of obtaining the result of the requested operation.

Deferred is not a "silver bullet" that allows a server to handle multiple requests in parallel, at a cost of less resources than with the system threads. Its advantages are manifested only when the server machine itself is not engaged in rating for the requested data and sends them to the processing of other services (network or local).

Figure 2 shows the basic organization of processing incoming requests using the *Deferred* approach, but in the case of processing the data received by the server. We see that it is no advantages over a simple request FIFO queue. In addition, resources are spent on maintaining the context of each request handlers and implementation of more complex logic. However, if the requests are handled by third-party services, the efficiency increases significantly. Figure 3 shows that the total processing time of three queries theoretically can be much smaller.

11. Our Used Approach

On the basis of the requirements use of server based on system processes calling not possible, so the choice was made between the use of lightweight threads and *Deferred* approach.

Since all tasks are processed on a single CPU, the main advantage of *Deferred* can not be fully realized. However, to justify the choice of server type was conducted two selected approaches tests.

12. Storage and Access to Data

Nowadays lot of information systems created specially to work under high-load uses non-relative databases named NoSQL solutions. Therefore, the data warehouse is not implemented using the relational approach, and it was chosen NoSQL solution. Each model is identified by a unique key that allows us to avoid the performance cost associated with processing complex SQL select queries.

To store patterns of user behavior, represented as serialized objects containing metadata model and a graph containing the model used, was selected one of the popular NoSQL systems: key-value store Redis (Lerner, 2011).

REDIS is an extension of the general idea of key-value, which supports not only the primitive operations get/save data by the key, but also supports more complex data structures without significant performance decreasing:

- Binary-safe strings;
- Hashes;
- Lists;
- The sets;
- Sort the set of (sorted sets).

Other important advantages of Redis is Sets, Lists with $O(1)$ operation complexity, push operation, "*lrange*" and "*ltrim*", server-side fast intersection between sets, are primitives that allow to model complex problems with a key value database (Lerner, 2011).

13. Operations on Graph of the Model

To implement the basic operations on graphs used library NetworkX (Hagberg *et al*., 2008), which is a separate module for the Python language for work with graphs, trees, networks, and other network structures.

Being completely Open Source project, NetworkX provides lot of useful features:

- Tools for studying the structure and dynamics of social, biological, and infrastructure networks;
- The development environment for quickly creating, testing, prototyping and implementation in various types of applications and research projects;

- Standard interfaces for use in popular programming languages such as C, C++, Fortran.
- Methods to easily import data from a large number of files in various formats;
- Classes for simple, oriented and weighted graphs;
- Save/load graphs to/from most popular file formats, storage of graphs;
- Built-in procedures for the creation of basic types of graphs;
- Methods for the detection of subgraphs, cliques and the K-core graph (maximal subgraph in which each vertex has at least the level of K).
- Obtaining such graph characteristics such as: degree of the vertices of the graph height, diameter, radius, length ways, the center, intermediate;
- Visualization of networks in the form of 2D and 3D graphs.

Capabilities of the library can freely operate a very large network structures level graph with 10 nodes and 100 millions of millions of arcs between them that even more redundant, since the graphs behaviors such amounts are not achieved. Given that the library is based on low-level Python data structure called "*dictionary of dictionaries*", the memory is consumed efficiently, graphs scale well, almost no dependent on the characteristics of the operating system on which the script is executed, and well suited to popular at this time scientific trend: "Analysis data from social networks, and graphs" (Hill and Dunbar, 2002).

14. Testing

As candidates was selected three popular Python servers:

- Twisted (Fettig, 2005);
- Tornado (Tornadoweb, 2011);
- Cyclone.

Of these, only Twisted directly supports *Deferred* approach. All three use by default system threads to process incoming data.

Methods of testing was that called modified server code, with added emulation of complex query processing, which uses 0.005 s of CPU time. To the target server creates a large number of concurrent requests, and was collected the final statistics of the server behavior.

Delayed response is implemented by following Python code:

```
import time
import random
rnd_delay = 0.005
time.sleep( rnd_delay )
```

In testing of the *Deferred* approach, delay in the code does not make sense, but used an additional service – available on a different port on the same local server machine. Used Apache web-server, which is also returns requested data after 0.005 seconds.

Self testing performed using the console program ApacheBench (ApacheBench, 1996). Used the following command:

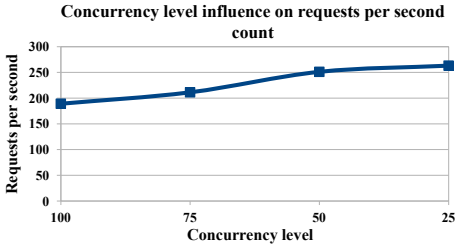

Fig. 4. Twisted, concurrency level VS requests per second.

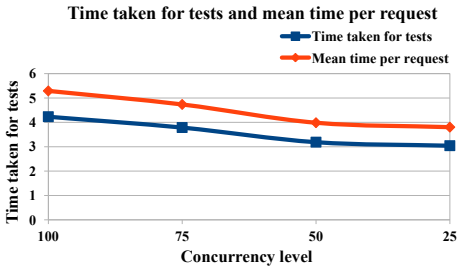

Fig. 5. Twisted, time taken for tests VS mean time per request without Defered.

Concurrency level	100.00	75.00	50.00	25.00
Time taken for tests	4.44	4.43	4.42	4.44
Requests per second	180.15	180.40	180.96	180.21
Time per request	555.10	415.73	276.31	138.73
Mean time per request	5.55	5.54	5.53	5.55

```
ab -n 800 -c 100 http://localhost:8007/
```

where:
ab – command, calling tests handler;
-n – total amount of requests;
-c – amount of parallel queries to server;
http://localhost:8007/ – target host-name for testing.

15. Twisted Server + Processing Without Deferred

Any (in the limit of experiment conditions) number of concurrent requests have been served in this configuration without changing the time for processing a single query. This shows that the 100 simultaneous threads created by modern operating systems handle without significant delay. Figure 4 shows a plot of the number of requests processed per second for a different number of concurrent connections to the server.

At Fig. 5 shows the processing of all requests for a different number of concurrent connections to the server.

16. Twisted Server + Deferred Processing

When the number of concurrent requests (Figs. 6, 7), the average processing time per request increases, but smaller amounts are shown the best results than using threads.

17. Tornado Server + Without Deferred

The result is similar to Twisted without using of *Deferred*, just a little more time on average is used for processing each request (Figs. 8, 9). It also shows the effective use system threads by server Twisted.

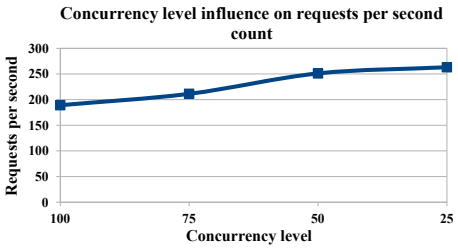

Fig. 6. Twisted, concurrency level VS requests per second with Deferred.

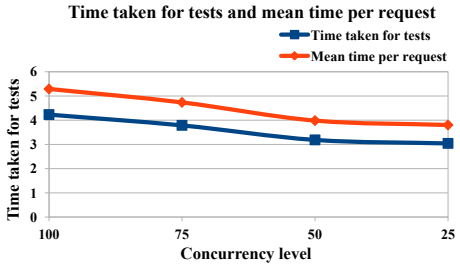

Fig. 7. Twisted, time taken for tests VS mean time per request with Deferred.

Concurrency level	100.00	75.00	50.00	25.00
Time taken for tests	4.23	3.79	3.19	3.04
Requests per second	189.09	211.28	251.07	263.10
Time per request	528.86	354.97	199.15	95.02
Mean time per request	5.29	4.73	3.98	3.80

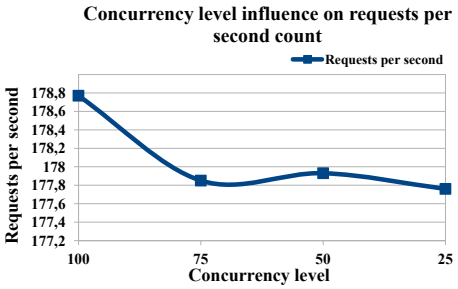

Fig. 8. Tornado, concurrency level VS requests per second without Deferred.

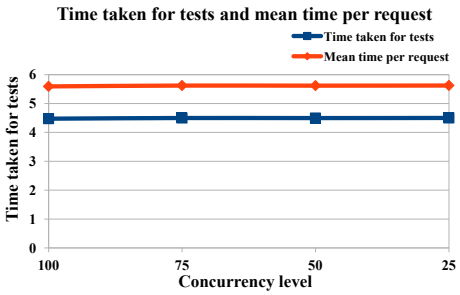

Fig. 9. Tornado, time taken for tests VS mean time per request without Deferred.

Concurrency level	100.00	75.00	50.00	25.00
Time taken for tests	4.48	4.50	4.50	4.50
Requests per second	178.77	177.85	177.93	177.76
Time per request	559.38	421.71	421.50	140.64
Mean time per request	5.59	5.62	5.62	5.63

18. Cyclone Server + Without Deferred

Cyclone server is based on the Twisted protocol, so that the behavior is similar to it naturally. Also, the result is predictable and that he showed little time-consuming (Figs. 10, 11) for processing each request as compared to Twisted.

19. Performance of PHP Server, what Emulates a Remote Service

To measure the influence of each request processing speed by PHP remote service emulation, was performed following testing.

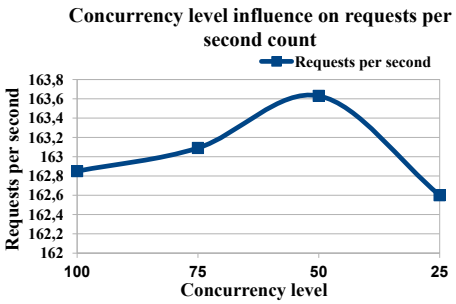

Fig. 10. Cyclone, concurrency level VS requests per second without Deferred.

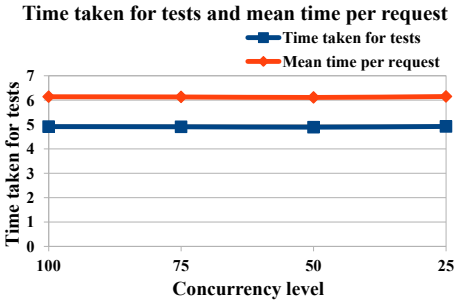

Fig. 11. Cyclone, time taken for tests VS mean time per request without Deferred.

Concurrency level	100.00	75.00	50.00	25.00
Time taken for tests	4.91	4.91	4.89	4.92
Requests per second	162.85	163.09	163.63	162.60
Time per request	614.07	459.88	305.57	153.75
Mean time per request	6.14	6.13	6.11	6.15

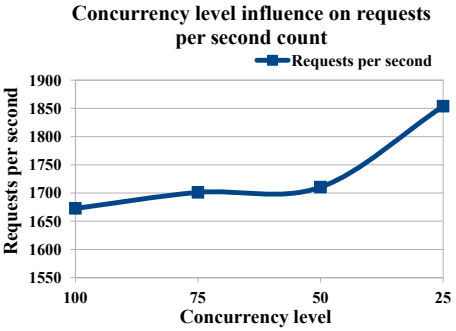

Fig. 12. Apache, concurrency level VS requests per second.

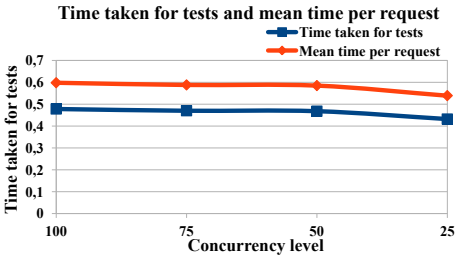

Fig. 13. Apache, time taken for tests VS mean time per request.

Concurrency level	100	75	50	25
Time taken for tests	0.47	0.47	0.46	0.43
Requests per second	1672.79	1701.16	1710.47	1853.89
Time per request	59.78	44.08	29.23	13.48
Mean time per request	0.59	0.58	0.58	0.53

However, Apache server proceeds such amounts of requests with confidence and with an increase of up to 100 simultaneous requests for the average processing time of each remained virtually unchanged (Figs. 12, 13).

20. Processing of Test Models

The above results are more appropriate for measuring of clean server's performance. However, for the closing to the real system conditions is important to measure results together

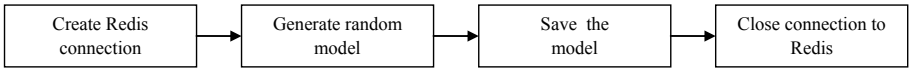

Fig. 14. Typical model processing flow.

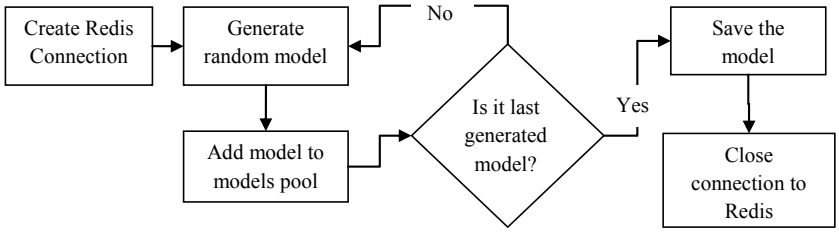

Fig. 15. Batch processing scheme.

with the loading and saving models operations. Have been done additional research to assess these parameters.

Initially was estimated time to create and maintain models in Redis directly from Python script without using the server as well as the generation of models will be presented separately from the server functioning.

A typical circuit implementation of this process is usually a linear operations flow what shown in Fig. 14, where each iteration loop adds random model consistently performs all operations for its creation and saving.

In the form of the procedure code is as follows:

```
r = redis.Redis(host='localhost', port=6379, db=0)
graps_in_test_min = 1
graps_in_test_max = 150
for y in xrange( graps_in_test_min, graps_in_test_max ):
    for x in xrange(0,y):
        graph_key_name = "users_group_\%s" \% str(x)
        print graph_key_name+' generated'
        rnd_graph = gen_random_graph(50)
        data_string = pickle.dumps(rnd_graph)
        r.set(graph_key_name, data_string)
```

In addition to the saving models using linear way, Redis also allows batch execution operations (Fig. 15), which allows to perform more effectively lot of transactions in one cycle without wasting resources on initialization and closing the connection to Redis at each iteration.

In the form of code, this scheme can be represented as follows:

```
r = redis.Redis(host='localhost', port=6379, db=0)
graps_in_test_min = 1
graps_in_test_max = 150

for y in xrange(graps_in_test_min,graps_in_test_max):
    pipe = r.pipeline()
    for x in xrange(0,y):
    graph_key_name = 'users_group_'+str(x)
```

```
        print graph_key_name+' generated'
        rnd_graph = gen_random_graph(50)
        data_string = pickle.dumps(rnd_graph)
        pipe.set(graph_key_name, data_string)
pipe.execute()
```

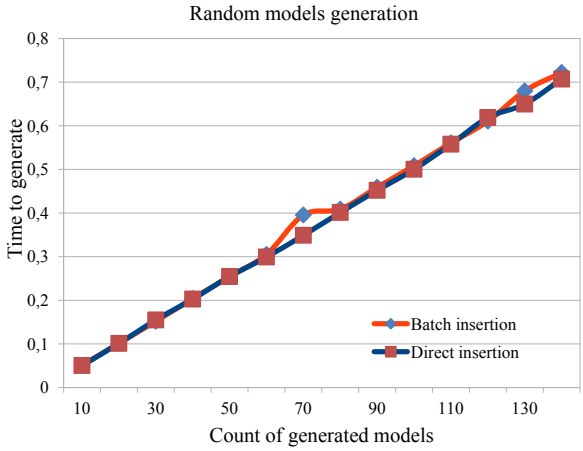

Fig. 16. Time to create a different number of models.

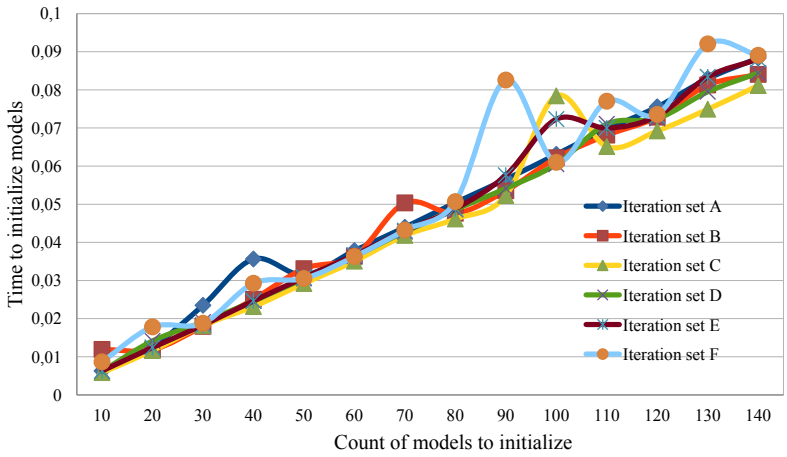

Fig. 17. Random models loading.

In Fig. 16 given time of creation and saving of 10–140 models with a random (but similar to the real) structure.

As can be seen in terms of the specifics of the problem domain (namely, because the Redis server is located on the same machine on which the test is performed, and all transactions with them is be made through memory) using batch processing requests does not give any gain in time (Fig. 16). Additionally it was estimated time to the random loading of the models without the use of servers, relying on the built-in to OS mechanisms for parallel computations optimizing (Fig. 17).

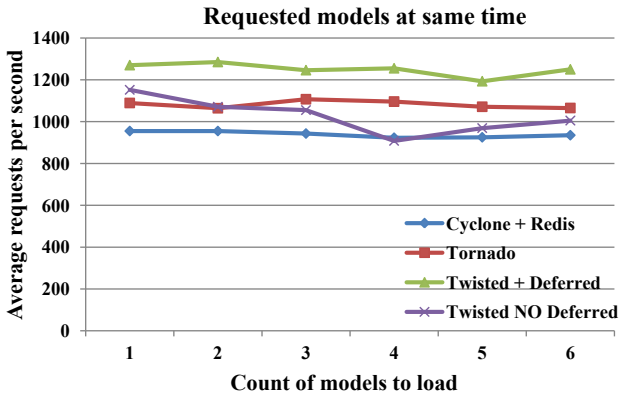

Fig. 18. Response time for different servers.

The data sets A, B and C show the three processes of loading 10–140 models. It is seen that in general the dynamics have a linear structure, while the bursts are random and probably depends from third-party processes in the system.

The data sets D, E and F also show the time taken to load the 10–140 models, but additionally after downloading it produced typical for algorithm used operations, by finding some characteristics of vertices and edges. It is seen that compared to the time model receiving the operations themselves take a little dismissive of resources.

Figure 18 shows the overall average response time by all used in this test servers, not with emulated artificial delay, but have a real models loading and carrying out a number of operations on it. It is evident that, as in the initial tests, *Deferred* approach shows better results.

21. Future Work

Architectural optimization can have a significant impact on the efficiency of the final module. The most important component is to translate all the code from functional to object-oriented style, what allows more efficient structure to make changes, new features implementation and support of system by third-party programmers. Also, object-oriented approach allows use of *JIT* compilation (Chang *et al*., 2011; PyPy, 2011; Perone, 2009) of Python code, which is designed to significantly accelerate the executable code as a script converts to a platform-specific binary format, which better uses features of the used server architecture. At the moment the possibility of JIT compilation for interpreted languages are widely studied in both academic and practice researchers. For example, a study of performance of the JIT compile PyPy (Perone, 2009) was showed that the optimization of Rastrigin function with many extremes

$$f(x) = 10n + \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i)$$

using genetic algorithms, with target number of generations equal to 40 000, took about 145 s versus 300 s for the nearest competitor (another method for Python code improvement) and with a maximum time of 550 s for the worst method.

Changes in the architecture to reduce the processing time of each transaction as well as the average response time is also not currently have a global character in comparison with the current architecture. A positive result can be given:

- Use a pool of connections to REDIS.
- Creating of the internal server caches, which, in view of the principle of locality (in relation to information systems), will save time for connection to the server REDIS.
- Micro optimizations at code-level for the calculation of each transaction metrics.

In addition, the obvious action aimed at increasing the productivity of the whole system is the use of more efficient hardware. In the future we plan to transfer to a cluster of computing power, available only through the external load balancer, evenly distributes incoming requests to internal nodes.

22. Conclusions

Experimental testing showed that using a single physical server that implements the processing of incoming transactions as well as direct operations on the model of user behavior, the complexity of the implementation of the *Deferred* approach may exceed the benefits derived from it compared to the use of system threads. However, the result shown with Deferred, is still better than using lightweight or heavy servers. Also, with growing of the number of concurrent processed by the system models in the future, this approach will provide a better average processing time of each transaction, so – as the benefits of Deferred approach more clearly shows with the increased load to the system.

References

Albert, R., Barabási, A.L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74, 47–97. arXiv:cond-mat/0106096.

ApacheBench (1996). Copyright 1996. Zeus Technology Ltd., Adam Twiss.

Bradshaw, R., Behnel, S., Seljebotn, D.S., Ewing, G., et al. (2013). *The Cython Compiler*. `http://cython.org`.

Chang, M., Mathiske, B., Smith, E., Chaudhuri, A., Bebenita, M., Gal, A., Wimmer, Ch., Franz, M. (2011). The impact of optional type information on JIT compilation of dynamically typed languages. In: *7th Dynamic Languages Symposium (DLS 2011)*, October 2011, Oregon, Portland.

Fettig, A. (2005). *Twisted Network Programming Essentials*. O'Reilly Media, 238 pp., ISBN:978-0-596-10032-2|ISBN10:0-596-10032-9.

Hagberg, A.A., Schult, D.A., Swart, P.J. (2008). Exploring network structure, dynamics, and function using NetworkX. In: Varoquaux, G., Vaught, T., Millman, J. (Eds.), *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Aug. 2008, Pasadena, CA, USA, pp. 11–15.

Hill, R., Dunbar, R. 2002. Social network size in humans. *Human Nature*, 14(1), 53–72.

Lerner, R.M. (2011). At the forge: redis. *Linux Journal*, 2010(197), September 2010, Article No. 5.

Lucks, J.B. (2008). *Python – All a Scientist Needs, Pycon*. arXiv:0803.1838v1 [q-bio.QM].

Millman, K.J., Aivazis, M. (2011). Python for scientists and engineers. *Computational Science & Engineering*, 13(2), 9–12, ISSN:1521-9615.

Osipov, P.A., Borisov, A.N. (2011). Abnormal action detection based on Markov models. *Automatic Control and Computer Sciences*, 41/2007–45/2011, ISSN 0146-4116 (print), 1558-108X (online), May 05, 2011.

Perone, C.S. (2009). Pyevolve: a Python open-source framework for genetic algorithms. ACM SIGEVOlution, 4(1), 12–20.

PyPy (2011). Final activity report: 28 month of European Union's funding phase PyPy. `http://codespeak.net/pypy/extradoc/eu-report/PYPY-EU-Final-Activity-Report.pdf`. Last accessed at 2011-12-06.

Samek, M. (2008), *Event-Driven Programming for Embedded Systems*. Newnes, 2008. ISBN-10:0750687061, ISBN-13:978-0750687065.

Tornadoweb (2011). Project web-page: `http://en.wikipedia.org/wiki/Tornado_(web_server)`. Last accessed at 2011-12-06.

P. Osipov Mg Sc Ing, PhD student, Institute of Information Technology, Riga Technical University. He received his master's diploma from Transport and Telecommunications Institute, Riga. His research interests include web data mining, machine learning and knowledge extraction.

A. Borisov is Professor of Computer Science in the Faculty of Computer Science and Information Technology at Riga Technical University. He holds a Doctor of Technical Sciences degree in Control in Technical Systems and the Dr. Habil. Sc. Comp. degree. His research interests include fuzzy sets, fuzzy logic, computational intelligence and bioinformatics. He has 210 publications in the area.

Atidėtojo metodo taikymo privalumai kritinio laiko uždaviniams

Pavel OSIPOV, Arkady BORISOV

Šiame straipsnyje aptariami sistemos organizavimo ir panaudojimo atvejai tipiškam vartotojo elgesio modelio konstravimui paskirstytos informacinės sistemos rėmuose, naudojant Python serverius. Daug dėmesio skiriama tyrimo proceso organizavimo analizei, naudojant Python kalbą visuose etapuose.

Peržiūrėtas Python serverių naudojimo galimybių taikymas įvairių uždavinių su griežtais veiksmų apribojimais gaunamų užklausų apdorojimui. Kartu su klasikiniu požiūriu, pasiūlytas naujas serverio struktūros variantas: atidėtasis, kuris neperžengia nagrinėjamos probleminės srities apribojimų