Балансировка нагрузки в системе Unihub на основе предсказания поведения пользователей.

Д.А. Грушин <grusin@ispras.ru>
Н.Н Кузюрин <nnkuz@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, дом 25

Аннотация. Разработанная в ИСП РАН программная система Unihub является облачной вычислительной системой типа SaaS и предоставляет пользователям возможность удаленной работы через Web-браузер с интерактивными графическими Linux-приложениями, запускаемыми в изолированных Docker контейнерах. Контейнеры имеют динамические требования к вычислительным ресурсам. Обычный способ размещения, при котором контейнеры распределяются равномерно по всем имеющимся в распоряжении серверам, приводит к неравномерной загрузке системы: некоторые сервера перегружены, а некоторые простаивают. В данной работе мы предлагаем собирать данные о поведении пользователей и характере работы различных приложений с целью прогнозирования создаваемой контейнерами нагрузки. Наши наблюдения показывают, что полученная информация позволяет равномернее загрузить имеющееся в распоряжении оборудование и увеличить предельную производительность системы в целом**.**

Ключевые слова: облачные системы типа SaaS. система Unihub. прогнозирование поведения пользователей

DOI: 10.15514/ISPRAS-2015-27(5)-2

Для цитирования: Грушин Д.А., Кузюрин Н.Н. Балансировка нагрузки в системе Unihub на основе предсказания поведения пользователей. Труды ИСП РАН, том 27, вып. 5, 2015 г., стр. 23-34. DOI: 10.15514/ISPRAS-2015-27(5)-2.

1. Введение

Одной из основных тенденций развития информационных технологий в настоящее время является массовое внедрение "облачных" вычислений. Важнейшим преимуществом для пользователей облачной инфраструктуры является возможность отказаться от необходимости самостоятельно содержать дорогостоящее вычислительное оборудование. За счет использования технологий виртуализации, одним набором физического

оборудования, облако способно удовлетворить различные требования большого числа пользователей. Облако также дает возможность "масштабирования с помощью кредитной карты" (ориг. "scale by credit card") – что означает незамедлительное предоставление по требованию дополнительных ресурсов на некоторое время за плату. При этом, потребности пользователя ограничиваются только его финансовыми возможностями, в отличие от физических ограничений при добавлении узлов в кластер или накладных расходов на содержание неиспользуемых ресурсов. Системы управления облачными инфраструктурами дают новые возможности для администраторов, так как включают в себя механизмы для мониторинга и управления огромными массивами физических серверов. Различают несколько моделей облачных вычислений:

1. Программное обеспечение как услуга (SaaS, англ. Software-as-a-Service) — модель, в которой пользователю предоставляется возможность использования программного обеспечения, работающего в облачной инфраструктуре и доступного из различных клиентских устройств, например, из браузера.
2. Платформа как услуга (PaaS, англ. Platform-as-a-Service) — модель, когда пользователь использует облачную инфраструктуру для размещения базового программного обеспечения для последующего размещения на нем новых или существующих приложений. В состав таких платформ входят инструментальные средства создания, тестирования и выполнения прикладного программного обеспечения — системы управления базами данных, среды исполнения языков программирования и др., предоставляемые облачным провайдером.
3. Инфраструктура как услуга (IaaS, англ. IaaS or Infrastructure-as-a-Service)предоставляется как возможность использования облачной инфраструктуры для самостоятельного управления ресурсами обработки, хранения, сетями и другими фундаментальными вычислительными ресурсами. Например, потребитель может устанавливать и запускать произвольное программное обеспечение, которое может включать в себя операционные системы и прикладное программное обеспечение. Потребитель может контролировать операционные системы, виртуальные системы хранения данных и установленные приложения, а также набор доступных сервисов (например, межсетевой экран, DNS).

Разработанная в ИСП РАН программная система Unihub [10] является облачной вычислительной системой типа SaaS и предоставляет пользователям возможность удаленной работы через Web-браузер с интерактивными графическими Linux-приложениями.

Популярность подобных систем, предоставляющих доступ к удаленным рабочим столам посредством тонкого клиента, в настоящее время сильно выросла. Это произошло благодаря повсеместному внедрению облачных

вычислений и доступности скоростного Интернет доступа [3]. При выполнении приложений в виртуальных рабочих столах на удаленных серверах, пользователи получают доступ к любому приложению из любого места и с любого устройства. Такая модель использования вычислительных ресурсов создает новую задачу оптимизации распределения нагрузки.

Этому направлению в настоящее время уделяется большое внимание (см. например [13-15]). В системе Unihub все задачи интерактивные, с динамическими требованиями к ресурсам. При появлении задачи планировщик не может знать когда она завершится и сколько ресурсов она будет потреблять в определенный момент времени. Если поместить несколько столов на один сервер, и приложения при этом начнут активно использовать процессор, то пользователи сразу заметят замедление работы, что нежелательно. Эта ситуация в точности соответствует работе нескольких приложений на одном компьютере - если вы запускаете много тяжеловесных приложений, то процессор делится между ними, и все приложения работают заметно медленнее.

Для оптимизации размещения рабочих столов планировщику необходима дополнительная информация. Источником такой информации может служить, например, история загрузки серверов за последнее время, количество запущенных приложений на каждом сервере, и т.д. Мы предлагаем собирать и хранить информацию о загрузке серверов вместе со статистикой запусков задачи и поведения пользователей в системе для оптимизации принятия решений планировщиком.

2. Управление вычислительными ресурсами в системе Unihub

Unihub запускает приложения в Docker контейнерах [11], которые могут размещаться в виртуальных машинах или на серверах (далее будем использовать термин сервер для обозначения обоих случаев).

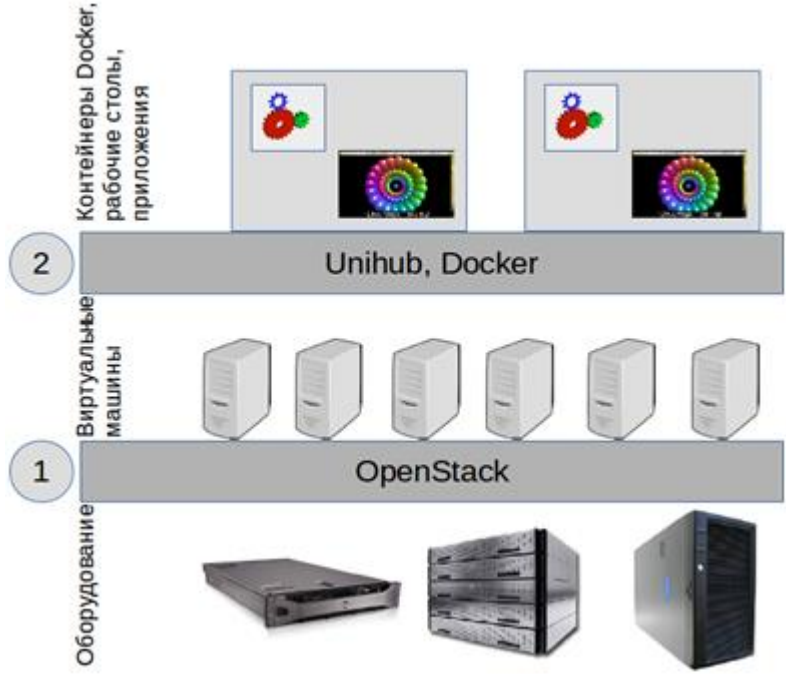
Контейнеры изолируют несколько экземпляров операционной системы Linux на одном сервере друг от друга и позволяют ограничивать процессорное время, память и другие ресурсы, потребляемые отдельным экземпляром.

Каждый контейнер содержит работающий процесс графического рабочего стола Xorg. Внутри рабочего стола пользователь может одновременно работать с несколькими графическими приложениями. В то время, когда рабочие столы продолжают постоянно работать на серверах Unihub, пользователь может отключаться и подключаться к системе в любое время и с любого совместимого устройства.

Работающие на сервере контейнеры контролируются Docker агентом. По умолчанию, если несколько контейнеров работают на одном сервере, то процессорное время разделяется между ними поровну. Однако, если другие контейнеры бездействуют, то один контейнер может использовать процессор полностью. Docker не дает возможности жестко ограничить использование

процессора одним контейнером, например, в размере 1/5, зато позволяет привязать контейнер к определенному процессорному ядру на все время работы контейнера.

Рассмотрим схему управления вычислительными ресурсами в системе Unihub.



На первом уровне вычислительным оборудованием управляет OpenStack. Он предоставляет вышестоящему уровню вычислительные серверы: как виртуальные машины, так и bare metal серверы.* Распределение серверов происходит на этапе инициализации системы, и управляется администратором. В дальнейшем распределении задач пользователей планировщик OpenStack не принимает участия.

На втором уровне планировщик системы Unihub создает и распределяет Docker контейнеры с рабочими столами пользователей по серверам. Контейнер использует квоты (ограничения) на количество разрешенной памяти и процессорного времени. Зная характеристики сервера и требования контейнеров, планировщик решает какое количество контейнеров следует разместить на данном сервере.

* bare metal сервер - это вычислительная система, в которой виртуальная машина устанавливается непосредственно на оборудование, а не поверх гостевой операционной системы.

В системе Unihub все задачи интерактивные, с динамическими требованиями к ресурсам. При появлении задачи планировщик не может знать когда она завершится и сколько ресурсов она будет потреблять в определенный момент времени. Это означает, что, если мы поместим несколько контейнеров на один сервер, и контейнеры при этом начнут активно использовать процессор, то пользователи сразу заметят замедление работы своих приложений,[*] что нежелательно.

Исходя из этого, представляется логичным распределять контейнеры равномерно по всем имеющимся в распоряжении узлам. Таким образом, замедление работы должно будет происходить при общей перегрузке системы - превышении некоторого количества одновременно работающих пользователей. В таких ситуациях говорят о достижении предельной производительности системы и необходимости наращивания оборудования.

Однако, очевидно, что пользователи используют рабочие столы по-разному. Характер нагрузки, создаваемой двумя произвольными контейнерами будет различаться. Это означает, что, разместив одинаковое количество контейнеров на каждом сервере, возникнет ситуация при которой некоторые сервера будут перегружены, а некоторые будут простаивать.

В данной ситуации, оптимальным решением является миграция работающего контейнера на менее загруженный сервер. В последней версии Docker возможность миграции контейнера появилась [12], однако все еще находится на стадии разработки.

Если миграция невозможна, и контейнер должен быть размещен только один раз и работать до конца на одном узле, необходимо следить за тем, чтобы предсказать нагрузку и распределить ее по серверам равномерно. Это значит, планировщику необходима дополнительная информация. Источником такой информации может служить, например, история загрузки серверов за последние несколько минут, количество запущенных приложений на каждом сервере, и т.д. Мы предлагаем использовать статистику запусков задачи и поведения пользователей в системе для оптимизации принятия решений о размещении контейнеров.

Рассмотрим пример работы контейнера в системе Unihub.

1. Пользователь авторизуется в системе.
2. Некоторое время пользователь работает с информационными материалами и не запускает приложений.
3. Пользователь запускает приложение X и некоторое время активно использует его.
4. Пользователь прекратил работать с приложением, но не закрыл его. Приложение работает, но не создает нагрузки.
5. Пользователь вышел из системы.

[*] QoE - quality of experience, качество восприятия.

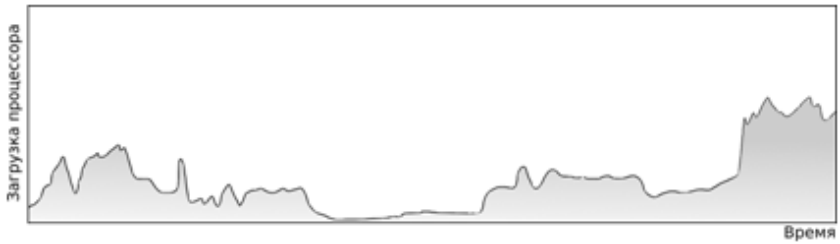
Учитывая то, что пользователи регулярно пользуются системой, данная модель поведения будет с некоторыми вариациями повторятся. Эта информация может быть использована планировщиком. Например, как только пользователь зашел в систему, планировщик уже может предсказать его дальнейшие действия. При создании контейнера планировщик также будет обладать информацией о статистике использования данного приложения как текущим, так и другими пользователями. Очевидно, редко используемые контейнеры можно группировать плотнее и изолировать их от более активных контейнеров, тем самым не вызывая заметных пользователям замедлений в работе приложений.

Использование данных о поведении пользователей в распределенных вычислительных системах неоднократно рассматривалось разными авторами, например, для оптимизации потребления энергии [6], для задачи размещения данных [5], оптимизации расписаний в Grid [7], снижения латентности тонких клиентов [8], запуска виртуальных машин [4] и др.

Во всех работах подтверждается периодичность, характерная для создаваемой пользователями нагрузки. В системе Unihub мы также провели анализ поведения пользователей и выяснили, что больше половины постоянных пользователей работают с системой по расписанию. Можно выделить несколько групп пользователей, работающих одновременно с разной периодичностью.

2.1 Сбор данных и размещение контейнеров

Рассмотрим пример графика создаваемой контейнером нагрузки.



С момента создания контейнера на некотором промежутке времени измеряется загрузка процессора. Мы предлагаем вычислять три величины: L - средняя загрузка на всем интервале, B - средняя продолжительность интервала высокой загрузки, I - средняя продолжительность интервала бездействия (низкой загрузки). В качестве промежутка времени предлагается использовать сессию пользователя.

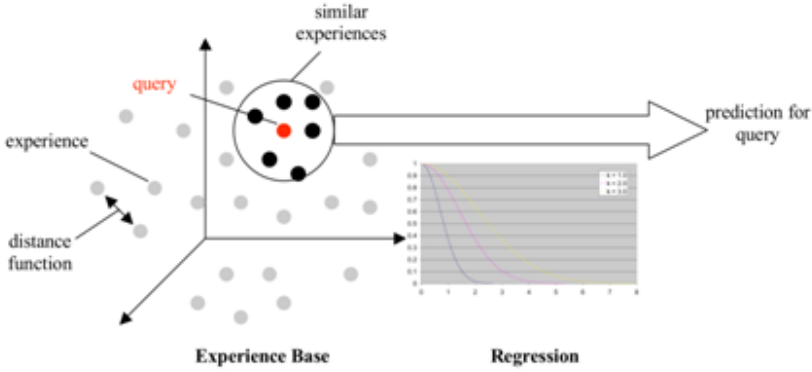
Вместе с указанными величинами сохраняется дополнительная информация:

1. идентификатор пользователя,
2. дата и время,
3. запущенные приложения в других контейнерах пользователя,

4. характеристики контейнера (квота, образ),
5. и др.

Располагая собранными данными, планировщику необходимо при размещении нового контейнера определить предполагаемый характер его нагрузки. Существует несколько способов решения данной задачи. Наиболее часто применяемыми на практике являются способы, основанные на машинном обучении [2]. В данной работе мы применили один из методов основанных на машинном обучении - instance-based learning (IBL), который иногда называется memory-based learning [1].

Данный метод работает следующим образом. В базе данных хранится информация об "опытах". В нашем случае это информация, связанная с историей загрузки контейнера. Опыт состоит из входных и выходных объектов. Входные объекты описывают условия, при которых опыт проводился, а выходные - то, что произошло при этих условиях. Каждый объект, как правило, состоит из имени и значения, где значением может быть простой тип: целое число, число с плавающей точкой, или строка. Запросом к базе является массив входных объектов, по которым мы хотим получить соответствующий опыт.



Близость двух опытов определяется функцией расстояния (distance function). Существует много вариантов вычисления данной функции [9].

Заключительным этапом формирования прогноза является построение массива полученных из базы близких опытов и доверительного интервала.

Получая таким образом значения L,B,I - предполагаемой средней загрузки и интервалов высокой и низкой активности, для размещения контейнера выбирается сервер, для всех контейнеров которого сумма значений предполагаемой средней загрузки в заданном интервале времени не превышала бы значения заданной константной величины L_max (например 0,8), и у которых отношение длины среднего интервала бездействия к средней длине интервала сильной загрузки (I/B) минимально.

В случае, когда планировщик не может определить предполагаемые значения загрузки для данного контейнера, размещение происходит на наименее загруженный за последнее время сервер.

Данные о принятом планировщиком решении также сохраняются. Если в процессе работы сервера величина загрузки превышает предполагаемую, то параметр L_max корректируется. Таким образом, планировщик будет обладать возможность адаптироваться к различным потокам задач.

2.2 Эксперименты

Для сравнения поведения указанных стратегий проводилось моделирование на основе собранной информации из работающей системы Unihub. Мы использовали ранее разработанную в ИСП РАН систему Gridme с некоторыми необходимыми изменениями для моделирования облачных систем.

Для сравнения была выбрана базовая стратегия - размещение контейнеров равным количеством по всем доступным серверам, при котором загрузка сервера не учитывается.

Второй стратегией мы выбрали размещение на наименее загруженный за последние сутки сервер.

Третья стратегия - размещение с учетом собранной дополнительной информации о загрузке контейнеров.

Эксперименты подтвердили, что дисбаланс при использовании базовой стратегии наступает очень быстро. Единственным плюсом базовой стратегии является простота реализации.

Стратегия размещения на наименее загруженный за последние сутки сервер показала лучший результат по отношению к базовой. Вероятность возникновения дисбаланса меньше, и зависит от периода времени за который суммируется загрузка сервера. Наилучшие результаты были показаны при значении интервала в 3 суток. Дальнейшее увеличение интервала приводило к ухудшению результатов.

Стратегия размещения с учетом собранной дополнительной информации показала лучший результат. В среднем, по сравнению с базовой стратегией удалось увеличить количество одновременно работающих контейнеров без ухудшения QoE на 12%.

3. Заключение

Система Unihub является облачной вычислительной системой типа SaaS, предоставляющей доступ к интерактивным приложениям. Основной характеристикой такой системы может являться величина QoE - качество восприятия. В настоящее время данный термин становится все более популярным. Довольно много исследований ведется по созданию новых методов определения качества восприятия пользователями того или иного сервиса или услуги, однако точных стандартизованных методик по

определению QoE, описывающих все случаи, пока не существует. Для системы Unihub величину QoE определяет отсутствие видимого замедления в работе приложений пользователя.

В системе Unihub пользователь, заходя через Web-браузер с любого совместимого устройства, может запустить несколько рабочих столов. Рабочий стол - это графическое окружение Linux, изолированное внутри Docker контейнера. В то время, как рабочие столы постоянно работают на серверах Unihub, пользователь может отключаться и подключаться к системе в любое время.

Такая модель характеризуется динамической нагрузкой. Учитывая, в настоящее время, невозможность миграции работающих контейнеров, основной задачей планировщика является равномерное распределение нагрузки по серверам. В случае дисбаланса значение QoE будет уменьшаться.

В данной работе мы предложили собирать и использовать информацию о характере активности пользователей для оптимизации размещения контейнеров. Наблюдения показали, что для создаваемой пользователями нагрузки характерна периодичность - больше половины постоянных пользователей работают с системой по расписанию, причем по этому показателю всех пользователей можно разделить на несколько групп.

Результаты моделирования предложенных алгоритмов показали, что использование информации о поведении пользователей позволяет поднять плотность упаковки контейнеров на одном сервере без значительных, заметных пользователю потерь производительности приложений. В сравнении с базовой стратегией (размещение контейнеров равным количеством по всем доступным серверам) удалось увеличить плотность упаковки контейнеров в среднем на 12%.

Мы планируем в дальнейшей работе исследовать степень влияния различных параметров на точность предсказания создаваемой контейнером загрузки. Необходимо также изучить устойчивость предсказания и, следовательно, эффективность балансировки, к резкому изменению поведения пользователей, например, при добавлении в систему большой группы новых пользователей.

Список литературы

[1]. Atkeson C., Moore A., Schaal S. Locally weighted learning. Artificial Intelligence Review. 1997. № 1-5 (11). С. 11–73.

[2]. Blum A. On-line algorithms in machine learning Springer, 1996. 306–325 с.

[3]. Deboosere L. [и др.]. Cloud-based desktop services for thin clients. Internet Computing, IEEE. 2012. № 6 (16). С. 60–67.

[4]. Jiang Y. [и др.]. ASAP: A self-adaptive prediction system for instant cloud resource demand provisioning 2011. 1104–1109 с.

[5]. Kavulya S. [и др.]. An analysis of traces from a production mapReduce cluster 2010. 94–103 с.

[6]. Nguyen T.-D. [и др.]. Prediction-based energy policy for mobile virtual desktop infrastructure in a cloud environment. Inf. Sci. 2015. № C (319). С. 132–151.

[7]. Smith W. Prediction services for distributed computing 2007. 1–10 с.
[8]. Suznjevic M., Skorin-Kapov L., Humar I. User behavior detection based on statistical traffic analysis for thin client services Advances in intelligent systems and computing. Под ред. Á. Rocha [и др.]., Springer International Publishing, 2014. 247–256 с.
[9]. Wilson D.R., Martinez T.R. Improved heterogeneous distance functions. J. Artif. Int. Res. 1997. № 1 (6). С. 1–34.
[10]. Unihub: Технологическая платформа программы университетский кластер. http://unihub.ru.
[11]. Docker: An open platform for distributed applications for developers and sysadmins. http://www.docker.com.
[12]. CRIU: A project to implement checkpoint/restore functionality for linux in userspace. http://criu.org/Docker.
[13]. Mohammadi F., Jamali S., Bekvari M., Survey on Job Scheduling algorithms in Cloud Computing, Int. Journal of Emergency Trends of Technology in Computer Science, 2014, v. 3, Issue 2, C. 151-154.
[14]. Tian W., Zhao E., Zhong V., Xu M., Jing C., A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters, Cloud computing and Intelligence Systems (CCIS), 2011, IEEE Int. Conference15-17 September 2011, C. 311-315.
[15]. Duy T.V.T., Sato Y., Inogushi Y., Performance evaluation of a green scheduling algorithm for energy savings in cloud computing, Parallel and Distributed Processing, Workshops and PhD forum, Atlanta, 19-23 April 2010, C. 1-8.

Load Balancing in Unihub SaaS System Based on User Behavior Prediction

D.A. Grushin <grushin@ispras.ru>
N.N. Kuzyurin <nnkuz@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. ISP RAS develops SaaS cloud computing system Unihub. The system provides possibility for users to work via Web-browser with interactive Linux-applications, running in isolated Docker containers. Each container runs Xorg process and variable amount of user applications. Containers share cpu time and have dynamical demands to computational resources. Conventional way of placement when containers are distributed uniformly among all servers can lead to bad result: some servers have too many running applications at some moment but others are almost empty.
Quality of Experience (QoE) is a measure of a customer's experiences with a service. In Unihub we evaluate QoE as an amount of visual slowdown of graphical applications.
In this paper we propose to collect information about users behavior and investigate how different applications work in order to predict containers load and provide this information to the scheduler.

Our observations showed that more then a halt of Unihub users work with the system on scheduled times, and depending on the schedule all users can be divided into several groups. The simulation results of the proposed algorithms have shown that the use of information about the user's behavior allows to run more containers on a given set of servers without significant loss of QoE.

Keywords: cloud computing SaaS systems, Unihub system. Users behavior prediction

DOI: 10.15514/ISPRAS-2015-27(5)-2

For citation: Grushin D.A., Kuzyurin N.N. Load Balancing in Unihub SaaS System Based on User Behavior Prediction. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 5, 2015, pp. 23-354 (in Russian). DOI: 10.15514/ISPRAS-2015-27(5)-2.

References

[1]. Atkeson C., Moore A., Schaal S. Locally weighted learning. Artificial Intelligence Review. 1997. № 1-5 (11). С. 11–73.
[2]. Blum A. On-line algorithms in machine learning Springer, 1996. 306–325 с.
[3]. Deboosere L. [и др.]. Cloud-based desktop services for thin clients. Internet Computing, IEEE. 2012. № 6 (16). С. 60–67.
[4]. Jiang Y. [и др.]. ASAP: A self-adaptive prediction system for instant cloud resource demand provisioning 2011. 1104–1109 с.
[5]. Kavulya S. [и др.]. An analysis of traces from a production mapReduce cluster 2010.С. 94–103.
[6]. Nguyen T.-D. [и др.]. Prediction-based energy policy for mobile virtual desktop infrastructure in a cloud environment. Inf. Sci. 2015. № С (319). С. 132–151.
[7]. Smith W. Prediction services for distributed computing 2007. С. 1–10.
[8]. Suznjevic M., Skorin-Kapov L., Humar I. User behavior detection based on statistical traffic analysis for thin client services Advances in intelligent systems and computing. Springer International Publishing, 2014. С. 247–256.
[9]. Wilson D.R., Martinez T.R. Improved heterogeneous distance functions. J. Artif. Int. Res. 1997. № 1 (6). С. 1–34.
[10]. Unihub: Technological platform for university cluster program. http://unihub.ru.
[11]. Docker: An open platform for distributed applications for developers and sysadmins. http://www.docker.com.
[12]. CRIU: A project to implement checkpoint/restore functionality for linux in userspace. http://criu.org/Docker.
[13]. Mohammadi F., Jamali S., Bekvari M., Survey on Job Scheduling algorithms in Cloud Computing, Int. Journal of Emergency Trends of Technology in Computer Science, 2014, v. 3, Issue 2, C. 151-154.
[14]. Tian W., Zhao E., Zhong V., Xu M., Jing C., A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters, Cloud computing and Intelligence Systems (CCIS), 2011, IEEE Int. Conference15-17 September 2011, C. 311-315.
[15]. Duy T.V.T., Sato Y., Inogushi Y., Performance evaluation of a green scheduling algorithm for energy savings in cloud computing, Parallel and Distributed Processing, Workshops and PhD forum, Atlanta, 19-23 April 2010, C. 1-8.