О задаче эффективного управления вычислительной инфраструктурой[1]

[1] *Д.А. Грушин <grushin@ispras.ru>*
[1,2] *Н.Н. Кузюрин <nnkuz@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский физико-технический институт, 141700, Московская область, г. Долгопрудный, Институтский пер., 9*

Аннотация. В настоящее время такие компании как Amazon, Google, Facebook, Microsoft, Yahoo! и другие управляют огромными дата-центрами из десятков тысяч узлов. Эти кластеры используются одновременно многими организациями и пользователями (облачная модель вычислений). Пользователи запускают задания, каждое из которых может состоять из одной или нескольких параллельных задач. Поток задач обычно представляет собой смесь: коротких, долгих, интерактивных, пакетных, и задач с различным приоритетом. Планировщик кластера решает, как разместить эти задачи на узлах, где они запускаются в виде процессов, контейнеров или виртуальных машин. Оптимизация размещения задач планировщиком позволяет улучшить степень использования узлов (machine utilization), сократить время отклика, сбалансировать нагрузку на части кластера, получить предсказуемую производительность приложений, повысить отказоустойчивость. Получить оптимальное размещение сложно. Для этого требуется решить задачу многокритериальной оптимизации, что требует времени. Это приводит к задержке при размещении очередной задачи, негативно сказывается на времени отклика и пропускной способности. С ростом размеров кластера и потока задач удовлетворить оба данных критерия становится сложным, и необходимо отдавать приоритет только одному. В данной статье мы рассмотрим архитектуру современных планировщиков и математические постановки задачи оптимизации.

Ключевые слова: оптимизация; планирование; виртуализация; облачные вычисления

DOI: 10.15514/ISPRAS-2018-30(6)-7

Для цитирования: Грушин Д.А., Кузюрин Н.Н. О задаче эффективного управления вычислительной инфраструктурой. Труды ИСП РАН, том 30, вып. 6, 2018 г., стр. 123-142. DOI: 10.15514/ISPRAS-2018-30(6)-7

[1] Работа выполнена при финансовой поддержке РФФИ, проект 17-07-01006.

1. Введение

С начала 2000 годов началось активное внедрение вычислительных кластеров[2] за счет использования стандартных компонент: обычных двух- или четырехпроцессорных рабочих станций (или персональных компьютеров) и коммуникационного оборудования (Myrinet, SCI, Fast Ethernet и др.) [4].

Основной частью спектра решаемых задач были параллельные MPI-программы [5]. Для решения таких задач кластер мог обходиться без системы управления (программа запускалась напрямую на всех узлах). Однако такой режим с трудом применим даже для одного пользователя. Для гибкого совместного использования дорогого вычислительного ресурса, система управления должна разделять кластер между поступающими от пользователей заданиями. Для этого предусмотрена очередь или набор очередей; необходимое количество узлов для запуска каждого задания выделяется таким образом, чтобы максимально использовать вычислительные возможности кластера. Более подробный список основных функций, обеспечиваемых системами управления, был впервые сформулирован в работе [6] и, позднее, в [7].

С развитием сетевых технологий стала набирать популярность архитектура Grid. Grid позволяет совместно использовать вычислительные ресурсы, которые принадлежат различным организациям и которые могут быть расположены в различных, географически разделенных административных областях. В Grid могут объединяться разнородные вычислительные ресурсы: персональные компьютеры, рабочие станции, кластеры и супер-компьютеры [8].

С появлением технологии виртуализации в процессорах и реализацией поддержки в операционных системах вычислительная инфраструктура начала стремительно изменяться. Между аппаратным и прикладным уровнями появился уровень виртуализации. Начался бум облачных вычислений. Теперь выделить для запуска программы набор “узлов” с одинаковым окружением стало значительно проще. Однородные кластеры стали превращаться в наборы разнородных серверов (в большей степени по причине обновления оборудования), сильно увеличиваясь в размерах, при этом спектр решаемых задач значительно расширился.

К сожалению, в настоящее время только небольшое число ученых имеет доступ к большим кластерам, и большинство работ на тему оптимизации вычислений сфокусировано на получении оптимального размещения в рамках ограниченных ресурсов. В реальности, задача оптимизации для конкретного кластера имеет множество возможных решений, относительная полезность каждого из которых зависит от характеристик потока задач и целей оператора кластера.

[2] Кластер определяется как множество рабочих станций (узлов), связанных коммуникационной средой и способных работать как единое целое за счет дополнительного программного обеспечения, называемого системой управления кластером.

Поток задач в большинстве случаев разнородный и на практике алгоритмы планировщика требуют ручной настройки для достижения хорошего качества размещения. С ростом размеров кластера данная задача становится трудновыполнимой.

Для принятия решения о размещении очередной задачи планировщику требуется время. Чем сложнее решение, тем это время больше. Для современных планировщиков оно находится в интервале от нескольких сотен миллисекунд до нескольких секунд. Очевидно, с ростом потока задач данная задержка будет приводить к увеличению времени отклика и времени ожидания, простою вычислительных ресурсов, создавая обратный эффект.

Конечно, проблема масштабирования систем управления и планировщиков больших кластеров не касается большинства пользователей. Наиболее простой способ масштабирования - это покупка необходимого количества ресурсов у облачного провайдера (Amazon Web Services, Google или Microsoft). Однако, масштабирование может быть важно и для небольших кластеров, если поток состоит из большого количества коротких интерактивных задач [9-10].

Также важно отметить, что качественное размещение экономит реальные деньги. Это важно как для владельцев больших систем (в Google отмечали, что их система планирования позволила избежать траты миллиардов долларов на строительство нескольких дата-центров [11]), так и для небольших организаций, где потеря нескольких сотен долларов в месяц из-за недогруженных виртуальных машин имеет значение.

Следует подчеркнуть, что оптимизация размещения задач на группах кластеров в различных моделях очень тесно связана с проблемами упаковки.

Эти вопросы будут рассмотрены далее для различных моделей.

2. Задача оптимизации размещения для однородного MPI кластера

Для параллельной программы необходимо выделить заданное количество процессорных ядер кластера в эксклюзивное пользование. В противном случае (две задачи используют одно ядро – overbooking) производительность сильно снижается, что приводит к непредсказуемому увеличению времени работы задач [12].

Узлы однородного кластера одинаковы, таким образом, кластер можно охарактеризовать количеством узлов (размер кластера).

Для этого нового класса задач в теории расписаний была предложена интересная геометрическая интерпретация в виде задачи упаковки последовательности прямоугольников в полубесконечную полосу. При этом ширина полосы соответствовала размеру кластера, а размеры прямоугольников соответствовали требованиям числа процессоров для задачи и времени исполнения. Эти модели очень тесно связаны [13-17].

Задача упаковки прямоугольников в полосы ставится следующим образом. Имеется набор различных прямоугольников и вертикальных (полубесконечных) полос. Стороны прямоугольников параллельны сторонам полос. Требуется упаковать прямоугольники в полосы без наложений и вращений (ортогональная ориентированная упаковка), оптимизируя заданные критерии (максимальную высоту заполнения, плотность заполнения, число прямоугольников в заполнении и другие). Эта модель очень тесно связана с оптимизацией выполнения параллельных задач на группах кластеров. Нам удалось получить целый ряд интересных и не имеющих аналогов результатов [18-22].

На практике наиболее распространенным способом распределения задач на кластере является алгоритм Backfill, когда для заполнения “пустых мест” в расписании используются задачи не с начала очереди [23]. При этом время работы задачи неизвестно, либо задается пользователем. В последнем случае оказывается, что время работы пользователи задают с большой положительной погрешностью, так как система принудительно завершает задачи по истечении выделенного времени [24]. В [25] было предложено не использовать время задаваемое пользователем, а предсказывать его на основе истории запуска (журнала) задач.

Задача упаковки прямоугольников в применении к вычислительному кластеру оптимизирует плотность размещения задач. В общем случае критериев оптимизации размещения задач на кластере может быть несколько. Наиболее распространены варианты:

- минимизация среднего времени ожидания задачи в очереди;
- минимизация максимального времени выполнения группы задач (makespan);
- максимизация пропускной способности - числа завершенных задач в единицу времени;
- минимизация простоев процессоров.

Для однородного кластера также интересной является задача снижения энергопотребления за счет гибких стратегий управления состоянием узлов (включения/выключения) и порядком выполнения заданий в очереди.

Основные способы, основаны на использовании программных средств, позволяющих повысить энергоэффективность кластера [26].

- Упаковка задач на минимальном количестве узлов и последующее выключение простаивающих узлов кластера.
- Перераспределение вычислительных заданий по времени при условии наличия многотарифной схемы оплаты электроэнергии (например, день-ночь). Тогда за счет повышения загрузки системы ночью днем вычислительная нагрузка будет снижена, и простаивающие компоненты вычислительной системы отключены. При этом общее количество потребленной электроэнергии не снижается, однако уменьшается ее

стоимость, что также рассматривается как повышение энергоэффективности.

- Программное управление производительностью компонентов вычислительной системы. Современные процессоры и оперативная память имеют возможность динамически изменять свою частоту и рабочее напряжение. Такой механизм носит название DVS (dynamic voltage and frequency scaling). Основной принцип данного механизма заключается в том, что при понижении напряжения процессора время вычислений увеличивается, однако общее количество энергии, потраченной на вычисления, уменьшается.

Поступающие задачи обычно требуют различных уровней “качества обслуживания” (Quality of Service – QoS). В применении к размещению задач качество обслуживания определяется как время от постановки задачи в очередь до получения результатов пользователем.

На практике данный критерий является основным. В некоторых случаях, возможно незначительно снижать качество обслуживания получая при этом значительную оптимизацию по другому критерию. Однако, ни один планировщик не может иметь дело с широким диапазоном ограничений и требований задаваемых как пользователями, так и провайдерами ресурса (лицензионные и экономические ограничения, отказоустойчивость, перегрев, пул свободных адресов и др.) [27].

Данная проблема решается возможностью ручной настройки (администратор ресурса определяет степень важности критериев) и вариациями стратегий распределения, разработанных с учетом конкретных потребностей.

3. Grid, расширение задачи на несколько полос

С развитием Grid-вычислений появилась возможность управлять потоком задач распределяя их на различные группы кластеров. При этом кластеры могут иметь различное число процессоров.

В связи с различными размерами кластеров задача оптимизации усложнилась [28] и стала по существу двух-уровневой: сначала для задачи выбирается кластер, а потом оптимизация размещения на кластере выполняется одной из известных эвристик. Геометрическая модель упаковки в одну полосу была обобщена нами на случай нескольких полос различной ширины [18], [22], [29-32], [20], [15].

Большинство работ, связанных с темой оптимизации вычислений в Grid, исследуют влияние различных мета-алгоритмов как на первом уровне, так и на втором уровне (распределение на кластере) [33].

Одним из критериев оптимизации также может быть минимизация потребляемой энергии. Согласно статистике, большинство кластеров испытывает периодическую нагрузку – когда интенсивность потока задач различается в несколько раз в разное время суток. Это означает, что даже при

относительно плотной загрузке заданиями в среднем, существуют периоды, когда большая часть узлов кластера не выполняет заданий и простаивает [26].

Для одиночного кластера, загруженного неравномерным потоком задач, достаточным является отключение простаивающих узлов. При этом от планировщика требуется применять различные стратегии сжатия - размещения максимального количества задач на минимальном количестве узлов.

В системе из нескольких кластеров, находящихся под управлением одного менеджера ресурсов – брокера существует несколько возможностей для экономии электроэнергии (как в количественном смысле, так и в денежном - снижая стоимость энергии) [34].

- Различная энергоэффективность (отношение производительности к энергопотреблению) кластеров;
- Географическое положение кластеров. Стоимость энергии в разных регионах может существенно различаться. Отправляя задачи на кластер с более низкой ценой электроэнергии можно уменьшить общую стоимость. Стоимость энергии различается также в разное время суток, что даёт дополнительные возможности выбора если кластеры находятся в разных часовых поясах.

4. Виртуализация

Виртуализация позволяет нескольким виртуальным машинам, часто с различными операционными системами, работать изолированно на одной физической машине. Каждая виртуальная машина имеет свой собственный набор виртуального оборудования (процессор, память, сетевые интерфейсы, дисковое хранилище), на котором загружаются операционная система и приложения. Операционная система использует набор аппаратных средств и не знает о совместном использовании с другими гостевыми операционными системами, работающими на одной и той же физической аппаратной платформе. **Гипервизор** (основной программный компонент технологии виртуализации) управляет взаимодействием между вызовами гостевой операционной системы, виртуальным оборудованием и фактическим выполнением, выполняемым на базовом физическом оборудовании.

Технологии виртуализации появились еще в 1960-х годах прошлого века. General Electric, Bell Labs, IBM пытались решить задачи расширения размеров оперативной памяти, возможности исполнения нескольких программ и др. Результатом стали такие разработки как суперкомпьютер Atlas, IBM 7044 (M44), CP/CMS, Livermore Time-Sharing System, Cray Time-Sharing System и др. [35]

Новейшая история виртуализации делится условно на два периода. Первый начался примерно в 1998 году и продолжался несколько лет. В эти годы происходило активное распространение идей виртуализации параллельно с

разработкой практических технологий. Когда технологии достигли “критической массы”, примерно в 2004 году, начался второй период – активное внедрение.

Сегодня виртуализация - это ключевая технология облачных вычислений и управления современным центром обработки данных. *Облачные вычисления* помогают решать проблемы с масштабируемостью, безопасностью и управлением глобальной ИТ-инфраструктурой, в то же время эффективно снижая затраты [36].

В отличие от классической виртуализации, контейнеризация в Linux – это технология виртуализации на уровне операционной системы, которая предлагает “легкую” виртуализацию. **Контейнеры** часто называют легковесными виртуальными машинами, однако, контейнер не является виртуальной машиной. Группа процессов, которая работает как контейнер, имеет свою собственную корневую файловую систему, но разделяет ядро с операционной системой хоста.

Система LXC (Linux Containers) - была добавлена в ядро Linux в 2008 году. LXC использует комбинацию таких функций ядра, как cgroups (позволяет изолировать и отслеживать использование ресурсов) и пространства имен (позволяют разделять группы так, чтобы они не могли “видеть” друг друга) [37].

Docker, появившийся несколько позже, позиционировался, как инструмент для упрощения работы по созданию и управлению контейнерами. Изначально Docker использовал LXC, затем была разработана библиотека под названием libcontainer. Docker сделал контейнеры доступными для обычного разработчика и системного администратора путем упрощения процесса и стандартизации интерфейса. Контейнеры Docker упаковывают программное обеспечение в полную файловую систему, в которой содержится все необходимое для запуска: код, среда выполнения, системные инструменты и системные библиотеки [38].

4. Различные архитектуры современных планировщиков.

Управление кластером и распределение задач являются ключевым компонентом современного дата-центра. Статья разработчиков Google [2] и книга [39], описывают архитектуру и задачи, решаемые современным планировщиком.

Планировщик Borg обеспечивает эффективное использование ресурсов за счет управления доступом, упаковки задач, наложению (over-commitment) и использованию контейнеров.[3]

[3]В оригинале “изоляция на уровне процесса”. Borg начал разрабатываться до активного распространения контейнеризации и виртуализации.

Он поддерживает приложения с высокой степенью доступности (high-availability applications) и использует эвристики распределения, уменьшающие время восстановления приложения и снижающие вероятность одновременного отказа копий приложения.

Borg предоставляет пользователям декларативный язык спецификации заданий, интеграцию сервисов имен, мониторинг работы в режиме реального времени и инструменты для анализа и моделирования поведения системы.

Borg – пример планировщика с *монолитной архитектурой*. Единственный процесс планировщика работает на одной машине и назначает задачи на остальные машины в кластере. Весь поток задач обрабатывается одним планировщиком (также планировщики с данной архитектурой можно называть **централизованными**). Это просто, универсально и позволяет применять сложные стратегии распределения.

Например, планировщики Paragon [40], Quasar [41], Mage [42] используют технологии машинного обучения для исключения негативного взаимного влияния потоков задач друг на друга.

Если решение о размещении очередной задачи требует сложных вычислений, то возникает задержка. Чем больше размер кластера и интенсивнее поток задач, тем больше задержка. Это накладывает ограничения на возможности использования таких планировщиков. Частичное решение проблемы было предложено авторами системы Firmament [43].

Одной из стратегий оптимизации, доступной централизованным планировщикам является консолидация задач без ухудшения качества обслуживания [44-45]. Это очень эффективная стратегия. В определенных случаях оптимизация от контролируемой консолидации может достигать сотен процентов [46].

Большинство кластеров выполняют задачи разного типа, требующих различных стратегий распределения. Уместить и поддерживать реализацию для каждого типа задач в одном планировщике, который обрабатывает смешанные (гетерогенные) рабочие нагрузки, может быть сложной задачей.

Авторы планировщика Mesos [47] поняли, что динамическое разделение кластера между различными типами потоков задач (Hadoop, Spark, TensorFlow) является ключевым фактором обеспечения эффективного использования вычислительных реурсов.

Планировщики Mesos и Yarn [48] имеют **двухуровневневую архитектуру**. На первом уровне ресурсы разделяются между планировщиками среды выполнения приложений, на втором – данные планировщики распределяют задачи.

Такая схема дает возможность использовать более эффективные стратегии управления вычислительными ресурсами в зависимости от специфики запускаемых приложений.

Однако, разделение процесса принятия решения о размещении задачи в двухуровневой архитектуре имеет также некоторые недостатки, так как планировщик лишается преимущества “всевидения”. В качестве последствий можно указать следующее.

- **Прерывание задач** (preemption) становится труднореализуемым. Ресурсы, занимаемые задачами, не видны планировщику первого уровня и он не может принять решение о замене задачи. Для реализации этого механизма необходимо передавать информацию или правила политики прерывания в планировщик среды выполнения приложений.
- **Интерференция** различных заданий (workloads), приводящая к потере производительности, так как планировщик может не знать “соседей” по серверу.
- **Сложность протокола** взаимодействия планировщиков двух уровней. При попытке решить вышеперечисленные проблемы возникает увеличение объема информации для обмена между планировщиками, что приводит к усложнению протокола их взаимодействия.

Планировщики **с разделяемым состоянием** призваны решить эти проблемы с помощью полу-распределенной модели. Состояние кластера независимо меняется планировщиками второго уровня. Если происходит конфликт, то изменение отменяется. Примерами являются планировщики Omega [49], Apollo [9], Nomad [50].

В системах с **распределенным состоянием** планировщики полностью независимы и не взаимодействуют между собой. Каждый планировщик работает со своей локальной копией состояния кластера. Задачи могут поступать на вход любому планировщику, который может размещать их на любых узлах кластера.

Таким образом, разделение ресурсов кластера между задачами является результатом случайности в решениях независимых планировщиков без какого-либо централизованного управления.

Данный принцип вероятностной балансировки нагрузки был описан в работе [51], а наиболее известным планировщиком с распределенным состоянием является Sparrow [52].

Планировщики с распределенным состоянием используют простую логику принятия решения о размещении задачи и, тем самым, могут обрабатывать поток заданий с большой скоростью.

Недостатком планировщиков с распределенным состоянием можно назвать невозможность реализовать строгий приоритет задач и справедливое распределение (fairness policies) [53]. Также, трудно исключить возможное взаимное влияние потоков задач друг на друга (интерференция).

Планировщики с **гибридной архитектурой** призваны решить проблемы и тех, и других разновидностей планировщиков за счет совмещения монолитной и распределенной архитектур. Мелкие задачи распределяет часть планировщика

с распределенной архитектурой, остальные - часть с монолитной. Гибридные планировщики описаны в работах Tarcil [54], Mercury [55], Hawk [56].

5. Размещение задач на примере планировщика Borg

Задание (возможно состоящее из нескольких задач) отправляется планировщику. Задание сохраняется в базе данных (распределенное хранилище), и составляющие задание задачи помещаются в очередь.

Очередь периодически сканируется планировщиком, который должен разместить задачу на подходящей машине. Сканирование очереди происходит в цикле по убыванию приоритета задач.

Алгоритм размещения состоит из двух этапов:

- поиск подходящих машин (на которых задача может быть выполнена),
- подсчет очков (scoring – процесс выбора машины из набора подходящих).

При поиске подходящих машин планировщик выбирает машины, которые удовлетворяют ограничениям задачи и имеют достаточно ресурсов (учитываются также ресурсы, занимаемые низкоприоритетными задачами, которые могут быть освобождены при необходимости).

При подсчете очков планировщик определяет значение функции $score(i,j)$ задачи j на машине i для всех машин. Выбирается машина с наименьшим значением.

Для вычисления значения $score$ планировщик учитывает требования задаваемые пользователем, однако, в большей степени, использует внутренние критерии оценки:

- минимизация количества и приоритета прерываемых (preempted) задач;
- выбор машин, на которых уже имеются необходимые для выполнения задачи данные;
- распределение/распыление (spreading) задач между энергетическими доменами и доменами безопасности; имеется ввиду попытка снижения пиковых энергетических нагрузок и снижения вероятности завершиния задачи из-за отказа оборудования;
- упаковка высокоприоритетных задач вместе с низкоприоритетными на одной машине, в случае всплеска нагрузки задачи с низким приоритетом могут быть прерваны, и освободившиеся ресурсы могут быть отданы оставшимся задачам с более высоким приоритетом.

Если выбранная машина не имеет достаточно ресурсов для запуска задачи, то планировщик начинает завершать (прерывать) задачи начиная с наименее приоритетных. Задачи отправляются обратно в очередь. Миграция и спящий режим не используются.

Наиболее важным критерием авторы указывают время запуска задачи - время от отправки задачи в очередь планировщика до запуска. Значения сильно разбросаны, медиана составляет 25 секунд. Из этого времени 80% занимает

установка необходимых пакетов ОС для работы задачи. Наиболее узким местом является запись на жесткий диск. Поэтому, для ускорения запуска задач планировщик старается размещать задачи на машины где уже присутсвуют необходимые пакеты.

6. Другие задачи планировщика.

Помимо эффективного распределения, любой планировщик должен уметь решать три основные задачи.

Подготовка (packaging). Программы, запускаемые планировщиком Borg, являются статически собранными бинарными файлами. Таким образом исключается проблема недостающих зависимостей в процессе выполнения. По сравнению с этим подходом, решение, предлагаемое Docker, гораздо универсальнее. Образы Docker позволяют собрать все необходимое для запуска программы в одном месте и использовать полученный образ на любом Docker сервере. В настоящее время стандартизацией формата образа контейнера занимается консорциум OCI [57].

Внедрение (deployment). Проблема, до сих пор не имеющая хорошего решения. Различные огранизации предлагают методы безопасного и надежного внедрения кода. В крупных компаниях существуют отдельные команды, занимающиеся разработкой инструментов автоматизации сборки и запуска новых версий ПО. Однако основные принципы внедрения ПО одинаковы. Необходимо остановить старую версию сервиса (программы) и запустить новую. При этом, желательно не допускать остановки обслуживания (обрыва соединений). В общем случае это делается путем направления всех поступающих запросов на новую версию сервиса, ожидая завершения обработки запросов старой версией.

Управления жизненным циклом (life-cycle). Планировщик должен контролировать выполнение сервиса, не допуская перебоев в обслуживании. Если программа завершается из-за внутренней ошибки, то планировщик может это определить и перезапустить процесс. В других случаях сервис может перестать отвечать на запросы из-за проблем с сетевым соединением. При отказе сервера необходимо перезапустить сервисы на других доступных серверах.

7. Заключение

Оптимизация управления является важнейшей задачей для современного вычислительного кластера и актуальна как для больших, так и для небольших систем. Наиболее распространена монолитная архитектура планировщика, которая позволяет реализовать сложные стратегии оптимизации и достичь хороших результатов. С другой стороны, при усложнении логики принятия решения о размещении задачи планировщик перестает масштабироваться. Решить проблему масштабирования призваны планировщики с

распределенной архитектурой, которые, в свою очередь, обладают некоторыми ограничениями.

Универсального решения в этом вопросе нет, и на практике зачастую приходится вручную настраивать различные параметры планировщика.

В ИСП РАН разработана система Fanlight – программная платформа для организации единой Web-среды для исследований, разработок и образования. Система может быть быстро развернута на имеющихся вычислительных ресурсах организации с последующей интеграцией в нее приложений, требующих поддержки аппаратного ускорения 3D графики. Вся последующая работа пользователя проводится через стандартный Web-браузер [58-59].

Мы разработали и интегрировали в систему Fanlight различные стратегии размещения контейнеров, позволяющие увеличить эффективность использования как CPU так и GPU устройств [60-61].

Список литературы

[1] Sinnen O. Task scheduling for parallel systems. John Wiley & Sons, 2007, 30 p.
[2] Verma A., Pedrosa L., Korupolu M. et al. Large-scale cluster management at google with borg. In Proc. of the Tenth European conference on computer systems, 2015, pp. 18:1–18:17.
[3] Ehrgott M. Multicriteria optimization. Springer, 2005, 320 p.
[4] Аветисян А., Грушин Д., Рыжов А. Системы управления кластерами. Труды ИСП РАН, том 3, 2002, стр. 39–62.
[5] Message P Forum. MPI: A message-passing interface standard. Technical Report. University of Tennessee, Knoxville, 1994, 228 p.
[6] Kaplan Joseph A., Nelson Michael L. A comparison of queueing, cluster and distributed computing systems. NASA Langley Technical Report, 1994, 49 p.
[7] Baker M., Fox G., Yau H. A review of commercial and research cluster management software. Northeast Parallel Architectures Center, 1996, 63 p.
[8] Foster I., Kesselman C. The grid: Blueprint for a new computing infrastructure. Morgan Kaufmann Publishers Inc., 1999, 675 p.
[9] Boutin E., Ekanayake J., Lin W. et al. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In Proc. of the 11th USENIX conference on Operating Systems Design and Implementation, 2014. pp. 285–300.
[10] Delgado P., Dinu F., Kermarrec A.-M. et al. Hawk: Hybrid datacenter scheduling. In Proc. of the 2015 USENIX annual technical conference, 2015.
[11] Schwarzkopf M. Cluster scheduling for data centers. Queue, vol. 15, no. 5, 2017.
[12] Herbein S., Dusia A., Landwehr A. et al. Resource management for running hpc applications in container clouds. Lecture Notes in Computer Science, vol. 9697, 2016. pp. 261–278.
[13] Ye D., Han X., Zhang G. Online multiple-strip packing. Theoretical Computer Science, vol. 412, no. 3, 2011, pp. 233–239.
[14] Hurink J. L., Paulus J. J. Online algorithm for parallel job scheduling and strip packing. Lecture Notes in Computer Science, vol. 4927, 2007. pp. 67–74.
[15] Zhuk S. On-line algorithms for packing rectangles into several strips. Discrete Mathematics and Applications, vol. 17, no. 5, 2007, pp. 517–531.

[16] Johnson D. S., Demers A., Ullman J. D. et al. Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM Journal on computing, vol. 3, no. 4, 1974, pp. 299–325.

[17] Garey M. R., Graham R. L., Johnson D. S. et al. Resource constrained scheduling as generalized bin packing. Journal of Combinatorial Theory, Series A, vol. 21, no. 3, 1976, pp. 257–298.

[18] Zhuk S., Chernykh A., Avetisyan A. et al. Comparison of scheduling heuristics for grid resource broker. In Proc. of the Fifth Mexican International Conference, 2004, pp. 388–392.

[19] Tchernykh A., Ramírez-Alcaraz J. M., Avetisyan A. et al. Two level job-scheduling strategies for a computational grid. Lecture Notes in Computer Science, vol. 3911, 2005. pp. 774–781.

[20] Tchernykh A., Schwiegelshohn U., Yahyapour R. et al. On-line hierarchical job scheduling on grids with admissible allocation. Journal of Scheduling, vol. 13, no. 5, 2010, pp. 545–52.

[21] Tchernykh A., Schwiegelshohn U., Yahyapour R. et al. Online hierarchical job scheduling on grids. In From grids to service and pervasive computing, Springer, 2008, pp. 77–91.

[22] Аветисян А.И., Гайсарян С.С., Грушин Д.А., Кузюрин Н.Н., Шокуров А.В. Эвристики распределения задач для брокера ресурсов Grid. Труды ИСП РАН, том 5, 2004, стр. 41–62.

[23] Baraglia R., Capannini G., Pasquali M. et al. Backfilling strategies for scheduling streams of jobs on computational farms. In Making Grids Work, Springer, 2008, pp. 103–115.

[24] Mu'alem A.W., Feitelson D.G. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 6, 2001, pp. 529–543.

[25] Nissimov A., Feitelson D.G. Probabilistic backfilling. Lecture Notes in Computer Science, vol. 4942, 2008. pp. 102–115.

[26] Иванников В.П., Грушин Д.А., Кузюрин Н.Н. и др. Программная система увеличения энергоэффективности вычислительного кластера. Программирование, том 36, no. 6, 2010, pp. 28–40.

[27] Baraglia R., Capannini G., Dazzi P. et al. A multi-criteria job scheduling framework for large computing farms. Journal of Computer and System Sciences, vol. 79, no. 2, 2013, pp. 230–244.

[28] Csirik J., Van Vliet A. An on-line algorithm for multidimensional bin packing. Operations Research Letters, vol. 13, no. 3, 1993, pp. 149–158.

[29] Кузюрин Н.Н., Поспелов А.И. Вероятностный анализ нового класса алгоритмов упаковки прямоугольников в полосу. Журнал вычислительной математики и математической физики, том 51, No. 10, 2011, стр. 1931–1936.

[30] Трушников М.А. Вероятностный анализ нового алгоритма упаковки прямоугольников в полосу. Труды ИСП РАН, том 24, 2013, стр. 457-468. DOI: 10.15514/ISPRAS-2013-24-21.

[31] Лазарев Д.О., Кузюрин Н.Н. Алгоритм упаковки прямоугольников в несколько полос и анализ его точности в среднем. Труды ИСП РАН, том 29, вып. 6, 2017, стр. 221–228. DOI: 10.15514/ISPRAS-2017-29(6)-13.

[32] Жук С.Н. О построении расписаний выполнения параллельных задач на группах кластеров с различной производительностью. Труды ИСП РАН, том 23, 2012, стр. 447-454. DOI: 10.15514/ISPRAS-2012-23-27.
[33] Tsai C.-W., Rodrigues J. J. Metaheuristic scheduling for cloud: A survey. IEEE Systems Journal, vol. 8, no. 1, 2014, pp. 279–291.
[34] Грушин Д.А., Кузюрин Н.Н. Энергоэффективные вычисления для группы кластеров. Труды ИСП РАН, том 23, 2012, стр. 433–46. DOI: 10.15514/ISPRAS-2012-23-26.
[35] Goldberg R.P. Survey of virtual machine research. Computer, vol. 7, no. 9, 1974, pp. 34–45.
[36] Magic Quadrant for Cloud Infrastructure as a Service, Worldwide. Available at: https://www.gartner.com/doc/3875999/magic-quadrant-cloud-infrastructure-service. Accessed 01.12.2018.
[37] Helsley M. LXC: Linux container tools. IBM devloperWorks Technical Library, 2009.
[38] Merkel D. Docker: Lightweight linux containers for consistent development and deployment. Linux Journal, no. 239, 2014.
[39] Barroso L. A., Clidaras J., Hoelzle U. The datacenter as a computer: An introduction to the design of warehouse-scale machines. Morgan & Claypool, 2013, 154 p.
[40] Delimitrou C., Kozyrakis C. Paragon: QoS-aware scheduling for heterogeneous datacenters. ACM SIGPLAN Notices, vol. 48, no. 4, 2013, pp. 77–88.
[41] Delimitrou C., Kozyrakis C. Quasar: Resource-efficient and qos-aware cluster management, ACM SIGPLAN Notices, vol. 49, no. 4, 2014, pp. 127–144.
[42] Romero F., Delimitrou C. Mage: Online and interference-aware scheduling for multi-scale heterogeneous systems. In Proc. of the 27th international conference on parallel architectures and compilation techniques, 2018, pp. 19:1–19:13.
[43] Gog I., Schwarzkopf M., Gleave A. et al. Firmament: Fast, centralized cluster scheduling at scale. In Proc. of the 12th usenix conference on operating systems design and implementation, 2016, pp. 99–115.
[44] Breitgand D., Epstein A. Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. In Proc. of the IEEE INFOCOM, 2012. pp. 2861–2865.
[45] Wang M., Meng X., Zhang L. Consolidating virtual machines with dynamic bandwidth demand in data centers. In Proc. of the IEEE INFOCOM, 2011. pp. 71–75.
[46] Urgaonkar B., Shenoy P., Roscoe T. Resource overbooking and application profiling in shared hosting platforms. In Proc. of the 5th symposium on operating systems design and implementation, 2002. pp. 239–254.
[47] Hindman B., Konwinski A., Zaharia M. et al. Mesos: A platform for fine-grained resource sharing in the data center. In Proc. of the 8th USENIX conference on Networked systems design and implementation, 2011. pp. 295-308.
[48] Vavilapalli V. K., Murthy A. C., Douglas C. et al. Apache hadoop yarn: Yet another resource negotiator. In Proc. of the 4th annual symposium on cloud computing, 2013, pp. 5:1–5:16.
[49] Schwarzkopf M., Konwinski A., Abd-El-Malek M. et al. Omega: Flexible, scalable schedulers for large compute clusters. In Proc. of the 8th ACM European conference on computer systems, 2013. pp. 351–364.
[50] Scheduling in Nomad. Available at: https://www.nomadproject.io/docs/internals/scheduling.html. Accessed 01.12.2018.

[51] Mitzenmacher M. The power of two choices in randomized load balancing. IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 10, 2001, pp. 1094–1104.
[52] Ousterhout K., Wendell P., Zaharia M. et al. Sparrow: Distributed, low latency scheduling. In Proc. of the twenty-fourth ACM Symposium on operating systems principles, 2013, pp. 69–84.
[53] Fagin R., Williams J. H. A fair carpool scheduling algorithm. IBM Journal of Research and development, vol. 27, no. 2, 1983, pp. 133–139.
[54] Delimitrou C., Sanchez D., Kozyrakis C. Tarcil: Reconciling scheduling speed and quality in large shared clusters. In Proc. of the sixth ACM Symposium on cloud computing, 2015, pp. 97–110.
[55] Karanasos K., Rao S., Curino C. et al. Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In Proc. of the USENIX annual technical, 2015. pp. 485–497.
[56] Delgado P., Dinu F., Kermarrec A.-M. et al. Hawk: Hybrid datacenter scheduling In Proc. of the USENIX annual technical conference, 2015. pp. 499–510.
[57] The Open Container Initiative. Available at: https://www.opencontainers.org/, accessed 01.12.2018.
[58] Аветисян А.А., Гайсарян С.С., Самоваров О.И. и др. Организация предметно-ориентированных научно-исследовательский центров в рамках программы университетский кластер. Труды конференции «Научный сервис в сети интернет: Суперкомпьютерные центры и задачи», 2010, стр. 213–5.
[59] Самоваров О.И., Гайсарян С.С. Архитектура и особенности реализации платформы unihub в модели облачных вычислений на базе открытого пакета openstack. Труды ИСП РАН, том 26, вып. 1, 2014, стр. 403-420. DOI: 10.15514/ISPRAS-2014-26(1)-17.
[60] Грушин Д.А., Кузюрин Н.Н. Балансировка нагрузки в системе Unihub на основе предсказания поведения пользователей. Труды ИСП РАН, том 27, вып. 5, 2015, стр. 23-34. DOI: 10.15514/ISPRAS-2015-27(5)-2.
[61] Грушин Д.А., Кузюрин Н.Н. Задачи оптимизации размещения контейнеров MPI-приложений на вычислительных кластерах. Труды ИСП РАН, том 29, вып. 6, 2017, стр. 229–244. DOI: 10.15514/ISPRAS-2017-29(6)-14.

On an effective scheduling problem in computation clusters

[1]*D.A. Grushin <grushin@ispras.ru>*
[1,2]*N.N. Kuzyurin <nnkuz@ispras.ru>*

[1]*Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2]*Moscow Institute of Physics and Technology, Dolgoprudnyj, Institutskij alley, Moscow region, 141700, Russia*

Abstract. At present, big companies such as Amazon, Google, Facebook, Microsoft, Yahoo! own huge datacenters with thousands of nodes. These clusters are used simultaneously by many users. The users submit jobs containing one or more tasks. Task flow is usually a mix of short, long, interactive, batch, and tasks with different priorities. Cluster scheduler decides on which server to run the task, where the task is then run as a process, container or a virtual

machine. Scheduler optimizations are important as they provide higher server utilization, lower latency, improved load balancing, and fault tolerance. Achieving good task placement is hard. The problem has multiple dimensions and requires algorithmically complex optimizations. This increases placement latency and limits cluster scalability. In this paper we consider different cluster scheduler architectures and optimization problems.

Keywords: optimization; scheduling; virtualization; cloud computing

DOI: 10.15514/ISPRAS-2018-30(6)-7

For citation: Grushin D.A., Kuzyurin N.N. On an effective scheduling problem in computation clusters. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 6, 2018, pp. 123-142 (in Russian). DOI: 10.15514/ISPRAS-2018-30(6)-7

References

[1] Sinnen O. Task scheduling for parallel systems. John Wiley & Sons, 2007, 30 p.
[2] Verma A., Pedrosa L., Korupolu M. et al. Large-scale cluster management at google with borg. In Proc. of the Tenth European conference on computer systems, 2015, pp. 18:1–18:17.
[3] Ehrgott M. Multicriteria optimization. Springer, 2005, 320 p.
[4] Аветисян А., Грушин Д., Рыжов А. Системы управления кластерами. Труды ИСП РАН, том 3, 2002, стр. 39–62.
[5] Message P Forum. MPI: A message-passing interface standard. Technical Report. University of Tennessee, Knoxville, 1994, 228 p.
[6] Kaplan Joseph A., Nelson Michael L. A comparison of queueing, cluster and distributed computing systems. NASA Langley Technical Report, 1994, 49 p.
[7] Baker M., Fox G., Yau H. A review of commercial and research cluster management software. Northeast Parallel Architectures Center, 1996, 63 p.
[8] Foster I., Kesselman C. The grid: Blueprint for a new computing infrastructure. Morgan Kaufmann Publishers Inc., 1999, 675 p.
[9] Boutin E., Ekanayake J., Lin W. et al. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In Proc. of the 11th USENIX conference on Operating Systems Design and Implementation, 2014. pp. 285–300.
[10] Delgado P., Dinu F., Kermarrec A.-M. et al. Hawk: Hybrid datacenter scheduling. In Proc. of the 2015 USENIX annual technical conference, 2015.
[11] Schwarzkopf M. Cluster scheduling for data centers. Queue, vol. 15, no. 5, 2017.
[12] Herbein S., Dusia A., Landwehr A. et al. Resource management for running hpc applications in container clouds. Lecture Notes in Computer Science, vol. 9697, 2016. pp. 261–278.
[13] Ye D., Han X., Zhang G. Online multiple-strip packing. Theoretical Computer Science, vol. 412, no. 3, 2011, pp. 233–239.
[14] Hurink J. L., Paulus J. J. Online algorithm for parallel job scheduling and strip packing. Lecture Notes in Computer Science, vol. 4927, 2007. pp. 67–74.
[15] Zhuk S. On-line algorithms for packing rectangles into several strips. Discrete Mathematics and Applications, vol. 17, no. 5, 2007, pp. 517–531.
[16] Johnson D. S., Demers A., Ullman J. D. et al. Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM Journal on computing, vol. 3, no. 4, 1974, pp. 299–325.

[17] Garey M. R., Graham R. L., Johnson D. S. et al. Resource constrained scheduling as generalized bin packing. Journal of Combinatorial Theory, Series A, vol. 21, no. 3, 1976, pp. 257–298.

[18] Zhuk S., Chernykh A., Avetisyan A. et al. Comparison of scheduling heuristics for grid resource broker. In Proc. of the Fifth Mexican International Conference, 2004, pp. 388–392.

[19] Tchernykh A., Ramírez-Alcaraz J. M., Avetisyan A. et al. Two level job-scheduling strategies for a computational grid. Lecture Notes in Computer Science, vol. 3911, 2005. pp. 774–781.

[20] Tchernykh A., Schwiegelshohn U., Yahyapour R. et al. On-line hierarchical job scheduling on grids with admissible allocation. Journal of Scheduling, vol. 13, no. 5, 2010, pp. 545–52.

[21] Tchernykh A., Schwiegelshohn U., Yahyapour R. et al. Online hierarchical job scheduling on grids. In From grids to service and pervasive computing, Springer, 2008, pp. 77–91.

[22] Avetisyan A.I., Gaissaryan S.S., Grushin D.A. et al. Scheduling heuristics for grid resource broker. Trudy ISP RAN/Proc. ISP RAS, vol. 5, 2004, pp. 41–62 (in Russian).

[23] Baraglia R., Capannini G., Pasquali M. et al. Backfilling strategies for scheduling streams of jobs on computational farms. In Making Grids Work, Springer, 2008, pp. 103–115.

[24] Mu'alem A.W., Feitelson D.G. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 6, 2001, pp. 529–543.

[25] Nissimov A., Feitelson D.G. Probabilistic backfilling. Lecture Notes in Computer Science, vol. 4942, 2008. pp. 102–115.

[26] Ivannikov V.P., Grushin D.A., Kuzyurin N.N., Pospelov A.I., Shokurov A.V. Software for improving the energy efficiency of a computer cluster. Programming and Computer Software, vol. 36, no. 6, 2010, pp. 327-336.

[27] Baraglia R., Capannini G., Dazzi P. et al. A multi-criteria job scheduling framework for large computing farms. Journal of Computer and System Sciences, vol. 79, no. 2, 2013, pp. 230–244.

[28] Csirik J., Van Vliet A. An on-line algorithm for multidimensional bin packing. Operations Research Letters, vol. 13, no. 3, 1993, pp. 149–158.

[29] Kuzjurin N.N., Pospelov A.I. Probabilistic analysis of a new class of rectangle strip packing algorithms. Computational Mathematics and Mathematical Physics, vol. 51, no. 10, 2011, pp. 1931–1936.

[30] Trushnikov M.A. Probabilistic analysis of a new rectangle strip packing algorithm. Trudy ISP RAN/Proc. ISP RAS, vol. 24, 2013, pp. 457-468 (in Russian). DOI: 10.15514/ISPRAS-2013-24-21.

[31] Lazarev D.O., Kuzjurin N.N. Rectangle packing algorithm into several strips and average case analisys. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 6, 2017, pp. 221–228 (in Russian). DOI: 10.15514/ISPRAS-2017-29(6)-13.

[32] Zhuk S. On parallel task scheduling on a group of clusters with different speeds. Trudy ISP RAN/Proc. ISP RAS, vol. 23, 2012, pp. 447-454. DOI: 10.15514/ISPRAS-2012-23-27.

[33] Tsai C.-W., Rodrigues J. J. Metaheuristic scheduling for cloud: A survey. IEEE Systems Journal, vol. 8, no. 1, 2014, pp. 279–291.

[34] Grushin D.A., Kuzjurin N.N. Energy effective computations on a group of clusters. Trudy ISP RAN/Proc. ISP RAS, vol. 23, 2012, pp. 433–46 (in Russian). DOI: 10.15514/ISPRAS-2012-23-26.
[35] Goldberg R.P. Distributed, low latency scheduling. In Proc. of the twenty-fourth ACM Symposium on operating systems principles, 2013, pp. 69–84.
[36] Magic Quadrant for Cloud Infrastructure as a Service, Worldwide. Available at: https://www.gartner.com/doc/3875999/magic-quadrant-cloud-infrastructure-service. Accessed 01.12.2018.
[37] Helsley M. LXC: Linux container tools. IBM devloperWorks Technical Library, 2009.
[38] Merkel D. Docker: Lightweight linux containers for consistent development and deployment. Linux Journal, no. 239, 2014.
[39] Barroso L. A., Clidaras J., Hoelzle U. The datacenter as a computer: An introduction to the design of warehouse-scale machines. Morgan & Claypool, 2013, 154 p.
[40] Delimitrou C., Kozyrakis C. Paragon: QoS-aware scheduling for heterogeneous datacenters. ACM SIGPLAN Notices, vol. 48, no. 4, 2013, pp. 77–88.
[41] Delimitrou C., Kozyrakis C. Quasar: Resource-efficient and qos-aware cluster management, ACM SIGPLAN Notices, vol. 49, no. 4, 2014, pp. 127–144.
[42] Romero F., Delimitrou C. Mage: Online and interference-aware scheduling for multi-scale heterogeneous systems. In Proc. of the 27th international conference on parallel architectures and compilation techniques, 2018, pp. 19:1–19:13.
[43] Gog I., Schwarzkopf M., Gleave A. et al. Firmament: Fast, centralized cluster scheduling at scale. In Proc. of the 12th usenix conference on operating systems design and implementation, 2016, pp. 99–115.
[44] Breitgand D., Epstein A. Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. In Proc. of the IEEE INFOCOM, 2012. pp. 2861–2865.
[45] Wang M., Meng X., Zhang L. Consolidating virtual machines with dynamic bandwidth demand in data centers. In Proc. of the IEEE INFOCOM, 2011. pp. 71–75.
[46] Urgaonkar B., Shenoy P., Roscoe T. Resource overbooking and application profiling in shared hosting platforms. In Proc. of the 5th symposium on operating systems design and implementation, 2002. pp. 239–254.
[47] Hindman B., Konwinski A., Zaharia M. et al. Mesos: A platform for fine-grained resource sharing in the data center. In Proc. of the 8th USENIX conference on Networked systems design and implementation, 2011. pp. 295-308.
[48] Vavilapalli V. K., Murthy A. C., Douglas C. et al. Apache hadoop yarn: Yet another resource negotiator. In Proc. of the 4th annual symposium on cloud computing, 2013, pp. 5:1–5:16.
[49] Schwarzkopf M., Konwinski A., Abd-El-Malek M. et al. Omega: Flexible, scalable schedulers for large compute clusters. In Proc. of the 8th ACM European conference on computer systems, 2013. pp. 351–364.
[50] Scheduling in Nomad. Available at: https://www.nomadproject.io/docs/internals/scheduling.html. Accessed 01.12.2018.
[51] Mitzenmacher M. The power of two choices in randomized load balancing. IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 10, 2001, pp. 1094–1104.
[52] Ousterhout K., Wendell P., Zaharia M. et al. Sparrow: Distributed, low latency scheduling. In Proc. of the twenty-fourth ACM Symposium on operating systems principles, 2013, pp. 69–84.

[53] Fagin R., Williams J. H. A fair carpool scheduling algorithm. IBM Journal of Research and development, vol. 27, no. 2, 1983, pp. 133–139.

[54] Delimitrou C., Sanchez D., Kozyrakis C. Tarcil: Reconciling scheduling speed and quality in large shared clusters. In Proc. of the sixth ACM Symposium on cloud computing, 2015, pp. 97–110.

[55] Karanasos K., Rao S., Curino C. et al. Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In Proc. of the USENIX annual technical, 2015. pp. 485–497.

[56] Delgado P., Dinu F., Kermarrec A.-M. et al. Hawk: Hybrid datacenter scheduling In Proc. of the USENIX annual technical conference, 2015. pp. 499–510.

[57] The Open Container Initiative. Available at: https://www.opencontainers.org/, accessed 01.12.2018.

[58] Avetisyan A.A., Gaissaryan S.S., Samovarov O.I. et al. Organization of scientific centers in univercity cluster program. In Proc. of the All-Russian Conference on Scientific service in Internet: Supercomputer centers and problems, 2010, pp. 213–215 (in Russian).

[59] Samovarov O.I., Gaissaryan S.S. Architecture and implementation details of Unihub platform in cloud computing architecture based on Openstack package. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 403-420 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-17.

[60] 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. 23–34 (in Russian). DOI: 10.15514/ISPRAS-2015-27(5)-2.

[61] Grushin D.A., Kuzyurin N.N. Optimization problems running mpi-based hpc applications. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 6, 2017, pp. 229–244 (in Russian). DOI: 10.15514/ISPRAS-2017-29(6)-14.