САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
КАФЕДРА КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ И МНОГОПРОЦЕССОРНЫХ СИСТЕМ

Кучумов Руслан Ильдусович

Магистерская диссертация

Исследование методов динамического масштабирования виртуального кластера

Направление: 02.04.02

Фундаментальная информатика и информационные технологии

Магистерская программа: ВМ.5502.2016

Вычислительные технологии

Научный руководитель:
кандидат физ.-мат. наук,
доцент
В. В. Корхов

Санкт-Петербург
2018

Содержание

1 Введение

Большое количество центров обработки данных по всему миру, включая различные научные и коммерческие организации, интегрируют облачные вычисления в свою инфраструктуру для размещения сервисов и выполнения вычислений. Преимущества использования облачных вычислений обсуждались не раз, например статья [1] дает неплохой обзор по этой теме. Для целей, поставленных в этой работе, основные преимущества использования облачных вычислений это их гибкая архитектура и возможность уменьшения стоимости обслуживания облачной вычислительной среды.

Рост облачных вычислений также привел к изменениям в сфере научных вычислений: они позволили ученым и исследователям запускать высокопроизводительные приложения не имея физической вычислительной инфраструктуры в своем распоряжении. Одним из ключевых факторов использования облачных вычислений может быть их экономическая модель, в которой пользователь платит только за те ресурсы, которые он использует (парадигма “pay-as-you-go”). При использовании этого подхода, арендовать виртуальные вычислительные ресурсы может быть выгоднее, чем покупать, поддерживать и обновлять вычислительное оборудование. В дополнении к уменьшению стоимости, облачная вычислительная среда предоставляет гибкость в выборе аппаратных и программных компонент, что обычно недоступно для традиционной вычислительной инфраструктуры, где вертикальное и горизонтальное масштабирование возможно только при замене вычислительной аппаратуры. Таким образом, основные преимущества использования облачной вычислительной среды это гибкость, снижение стоимости обслуживания и масштабируемость.

Парадигмы распределенных вычислений могут быть классифицирова-

ны на высокопроизводительные вычисления (high-performance computing, HPC), high-throughput computing (HTC) и многозадачные вычислительные системы (Many Tasks Computing, MTC), которые обычно разделяются по мере взаимодействия между вычислительными задачами и времени их выполнения. В HPC приложениях, задачи сильно-связаны и требуют большое количество вычислительных мощностей на короткий промежуток времени (часы или дни). С другой стороны, в HTC приложениях задачи слабо связанны и могут выполняться на порядки дольше HPC приложений: месяцами и годами. Парадигма MTC соединяет разрыв между HPC и HTC. Приложения MTC отличаются большим количеством задач с относительно коротким временем выполнения, производительность которых зависит скорее от пропускной способности хранилищ данных, чем от пропускной способности сети.

В этой работе будут рассматриваться HPC приложения для научных вычислений, последовательность разработки и использования которых обычно сводится к следующей. Группе ученых необходимо выполнить некоторые вычисления, например, провести имитационное моделирование. Для выполнения этой задачи эта группа ученых разрабатывает приложение собственными усилиями или пользуется услугами разработчиков. Реализованное приложение затем портируется и выполняется в кластерной вычислительной среде кластера. Одна и та же версия приложения обычно выполняется много раз подряд с различием во входных параметрах и наборах данных для обработки. При выполнении приложения в облачной вычислительной среде для каждого пользователя или приложения новый виртуальный вычислительный кластер должен быть создан и настроен самими пользователями или системными администраторами. В обоих случаях, это приводит к задержкам в достижении научных результатов и неоправданным осложнениям.

Кроме этого, ситуацию осложняет то, что специалисты не в IT областях

(например биологи и физики), не имеют и не должны иметь опыт опыта или умения подготовки задач для вычисления, работы с планировщиками задач и особенно настраивать вычислительный кластер. Вместо всех этих сложностей, они предпочтут оставить эти задачи системным администраторам и сфокусироваться на проблемах в своей области.

Для решения описанных выше проблем, было предложено реализовать программную систему, которая поможет пользователям выполнять HPC вычисления в облачной вычислительной среде. Она предоставит пользователям веб-интерфейс, позволяющий им указывать входные параметры задачи, отправлять задачу на выполнение и получить результат ее выполнения после завершения. Когда задача отправлена для выполнения, сервис должен запустить виртуальный кластер, настроить и подготовить задачу для выполнения и после ее выполнения, загрузить результаты ее работы в облачную директорию пользователя.

В этой программной системе детали создания кластера, его настройки и выполнения задачи скрыты от пользователя. Помощь системного администратора может понадобиться только для регистрации нового приложения и для обслуживания системы. Взаимодействие пользователя происходит только при вводе параметров задачи и при получении результатов.

Один из ключевых параметров запуска задач, за определение значения которого отвечает программная система, это количество требуемых вычислительных ресурсов или конфигурация кластера. В нее входит количество узлов, потоков в каждом узле и памяти. Пользователь, запускающий задачу и даже программист ее разработавший, не всегда могут с точностью определить при какой конфигурации кластера задача будет выполняться быстрее или какая конфигурация использует неоправданно много ресурсов. В данной работе предлагалось спроектировать метод автоматического определения оптимальной конфигурации кластера и разработать описанную выше программную систему, в которой можно будет применить этот

метод.

2 Постановка задачи

Целью этой работы является проектирование метода автоматического определения оптимальной конфигурации кластера и разработка программной системы для запуска вычислительных задач на облачных ресурсах, использующей спроектированный метод.

Для достижения цели необходимо было решить следующие задачи:

- исследовать существующие алгоритмы автоматического определения конфигурации виртуального кластера
- спроектировать метод, применимый для решения поставленной задачи
- исследовать существующие решения для запуска HPC приложений в облачной вычислительной среде
- спроектировать, разработать и протестировать программную систему для для запуска приложений в облачной вычислительной среде,
- протестировать приложения при различных конфигурациях кластера, используя программную систему
- протестировать сходимость спроектированного метода

К реализуемой программной системе были поставлены требования:

- получение запросов на выполнение вычислительных задач,
- создание и настройка виртуального вычислительного кластера,
- подготовка файлов входных данных задач,
- запуск задачи на созданном вычислительном кластере,

- выгрузка результатов задачи в облачное хранилище пользователя,
- отключение виртуального кластера,
- при последующих запусках задачи оптимизировать конфигурацию, кластера.

3 Обзор литературы

Есть несколько работ, целью которых было создание веб-интерфейса для планировщиков задач [2] и [3].

В первой работе для создания веб-интерфейса для планировщика HTCondor реализовали расширение для GridSAM – системы запуска и мониторинга вычислительных задач. GridSAM служит общим интерфейсом для ряда систем управления ресурсами и планировщиками (Condor, LSF, Sun Grid Engine) задач, поддерживающими стандарт описания вычислительных задач Job Submission Description Language (JSDL).

Во второй работе был создан веб-интерфейс для PBS планировщика задач. Его целью было упрощение работы пользователя при запуска задач. Веб-интерфейс сохраняют историю запусков задач, что позволяет перезапустить задачу “как есть” или с небольшими изменениями. Кроме этого, веб-интерфейс делает выбор вычислительного центра и запуск задач в нем прозрачным для пользователя.

Ключевое отличие от этих работ в том, что реализуемая в этой работе программная система может быть охарактеризована как общий интерфейс для различных планировщиков задач. У пользователя есть возможность только указывать только входные параметры параметры своего приложения и где его выполнять, при этом специфичные для планировщика параметры, такие как файл задачи или пути к файлам от пользователей скрыты.

Есть также публикации о системах, аналогичных разрабатываемой в этой работе, для управления виртуальными кластерами. “Virtual Cluster as a Service” [4] позволяет пользователям создавать виртуальный кластер для выполнения задач на простаивающих вычислительных ресурсах IT инфраструктуры университета под управлением OpenNebula. При работе с этой системой пользователи (в основном студенты) сначала создают виртуальный кластер из предварительно настроенных образов виртуальных машин и отправляют на него различные задачи для выполнения. После их завершения пользователь может запустить другие задачи или выключить виртуальный кластер. Кроме этого, при возникновении пиковых нагрузок, эта система может размещать часть запросов за счет ресурсов коммерческого облака Amazon Web Services.

“Dynamic Virtual Cluster” [5] – это похожая система, которая позволяет создавать виртуальный кластер для выполнения вычислительных задач. Принципиальное отличие в том, что в ней не используются облачные провайдеры, а виртуальные машины создаются напрямую с помощью гипервизора Xen. Задачи для выполнения на узлах кластера назначаются с помощью планировщика Moab. В отличие от предыдущей работы, виртуальный кластер создается специально для выполнения одной вычислительной задачи.

В “Virtual Organization Cluster” [6] виртуальный вычислительный кластер создается автоматически для выполнения задач, поступающих из грида. Аналогично предыдущей работе, программная система напрямую взаимодействует с гипервизором (KVM) для создания виртуальных машин. Для получения задач из грида и запуска их на узлах виртуального кластера используется планировщик HTCondor.

Аналогично перечисленным работам, в реализуемой программной системе создается кластер из виртуальных машин для запуска пользовательских задач. Но она предоставляет больше гибкости для выбора вычислительной

среды. Пользователь может выбрать на каких ресурсах запускать свою задачу, он может выбрать различные облачные провайдеры или использовать существующий физический вычислительный кластер.

Работа со схожей целью упростить процесс настройки виртуального кластера и запуска в нем HPC задач для не специалистов IT-индустрии была описана в [7]. Реализованная ими программная система позволяет запускать приложения на IaaS и PaaS облаках, предоставляет пользователям веб-интерфейс для запуска задач, для которых они указывают только входные параметры. Пользователи также могут выбирать существующие в системе приложения для запуска и регистрировать свои. Кроме того, что предложенная ими система была недоступна для использования, она не подходила, так как не использовала никакие подходы для оптимизации конфигурации виртуального кластера в зависимости от выполняемой задачи.

Ниже перечислено несколько работ, в которых авторы используют различные методы построения самонастраивающихся систем для автоматического определения конфигурации виртуального кластера, с целью выполнения различных критериев.

В работе [8] предложен алгоритм машинного обучения для определения того, какие типы виртуальных машин выбирать в коммерческом облаке, чтобы минимизировать их стоимость и при этом выполнить сроки завершения вычислительных задач. А именно рассматривается облачный провайдер Amazon Web Services, который позволяет пользователям выбрать либо арендовать виртуальные ресурсы по фиксированной цене, либо использовать свободные ресурсы (spot instances) со значительной скидкой (до 90%), но возможным их выключением через некоторое время. Для решения этой задачи авторами предложен алгоритм машинного обучения, который в зависимости от цен на ресурсы и время, требуемое на выполнение задачи, ранжирует набор стратегий выделения ресурсов и предлагает

наиболее оптимальную. Предложенный в этой работе алгоритм не подходит для решения поставленной задачи определения оптимальной конфигурации кластера, так как он не использует никакую информацию о свойствах выполняемой задачи, а оперирует только их потребностями ресурсов и ограничениями на время работы. Вместо построении модели зависимости оптимальной конфигурации от параметров задачи строится модель зависимости оптимальной стратегии выделения ресурсов от требований задачи и стоимости ресурсов.

В работе [9] рассматривается задача минимизации виртуальных ресурсов и их стоимости для выполнения последовательности вычислительных задач. На вход алгоритму подается последовательность задач, каждой задаче для выполнения могут требоваться различное количество и тип ресурсов. После завершения одной задачи, ресурсы могут быть повторно использованы последующими задачами. В работе показано, что описанная задача является NP-полной и предложен переборный алгоритм поиска оптимального множества ресурсов. Для решения поставленной задачи предложенный алгоритм тоже не подходит, так как он находит конфигурацию кластера, которая удовлетворит требованиям всех задач и при этом будет иметь минимальную стоимость. Как и в предыдущем случае, требования к ресурсам задачи остаются фиксированными.

4 Задача определения конфигурации кластера

Введем следующие обозначения:

- $y \in \mathbb{Y} \subset \mathbb{N}^N$ – вектор конфигурации кластера (количество узлов, потоков, памяти),
- $x \in \mathbb{R}^M$ – вектор параметров задачи любой конечной размерности с

непрерывными значениями,

- $t = t(x, y) \in \mathbb{R}$ – время выполнения задачи при параметрах x на конфигурации кластера y,
- $y^* = y^*(x) : t(x, y^*) \to min$ – оптимальная конфигурация кластера для параметров задачи x, при которой время выполнения задачи минимально

После ввода пользователем параметров задачи, программная система должна предложить оптимальную конфигурацию, при которой задача выполнится быстрее. Другими словами, задача состоит в определении регрессии x на y^*.

5 Метод определения конфигурации кластера

Вычислить точное значение $y^* = y^*(x)$ для фиксированного x на практике не всегда представляется возможным, так как задачу с параметрами x необходимо будет выполнить для всех возможных конфигураций кластера y. При этом, каждая задача может занимать ресурсы вычислительного кластера на неопределенное время. Ввиду этого, для определения y^* были выбраны методы случайного поиска, которые выдают последовательность оценок y'_n, сходящиеся к y^*.

О целевой функции $t = t(x, y)$ неизвестна никакая априорная информация: она может быть нелинейна, невыпукла, зашумлена, может не иметь производных. Кроме этого, ее “дорого” вычислять, так как ее вычисление инициализирует пользователь при отправке запроса на вычисление задачи.

Для задач оптимизации с такими особенностями целевой функции используют подход, называемый surrogate modeling [10]. Он используется обычно для моделирования сложных технических систем, где значимые

свойства не могут быть измерены напрямую, либо их измерение занимает много времени и ресурсов. Поэтому вместо проведения экспериментов создают вспомогательную модель, имитирующую поведение модели явления настолько точно, насколько это возможно, и при этом более простую в вычислении. В приложении к задаче оптимизации основная идея этого метода заключается в том, что вместо оптимизации целевой функции оптимизируется ее аппроксимация. Функция аппроксимации относительно проста для поиска экстремума, и его в некоторых случаях можно получить аналитически.

К таким методам относятся, к примеру, DONE [10], NEWUOA [11], метод байесовской оптимизации [12], в которых целевая функция аппроксимируется рядом Фурье, полиномами второго порядка или случайным гауссовским процессом соответственно.

Для построения модели регрессии $x \mapsto y^*$ можно воспользоваться методами многомерной нелинейной регрессии, но так как при случайном поиске для каждого x создается модель зависимости $t = t(x, y)$, то ее так же можно использовать и для определения $y^* = y^*(x)$. Таким образом, построив для возможных значений параметров задачи модели зависимости времени выполнения от конфигурации кластера, можно определять оптимальную конфигурацию кластера для каждого x.

5.1 Алгоритм DONE

Алгоритм DONE (data-based online nonlinear extremum-seeker) предназначен для решения задачи оптимизации:

$$\min_x f(x), x \in \mathbb{X},$$

$$f : \mathbb{X} \mapsto \mathbb{R}, \mathbb{X} \subset \mathbb{R}^n.$$

Он разработан для ситуаций, когда функция f нелинейна, невыпукла,

зашумлена, с неизвестными производными и сложна в вычислении. Такие функции обычно возникают при представлении результатов экспериментов или имитационного моделирования. В таких случаях, аналитическое выражение для производных получить невозможно и приближение с помощью конечных разностей также затруднительно из-за сложности вычисления значений и их зашумленности. Свойство вычисления в реальном времени (online) означает, что вычислительная сложность каждой итерации не растет при увеличении количества итераций и новое значение можно получить, не пересчитывая предыдущие.

Отличительная особенность алгоритма DONE в том, что для оптимизации целевой функции строится ее аппроксимация с помощью случайного ряда Фурье. У полученной аппроксимации существуют производные и она проста в вычислении. Для поиска ее минимума можно использовать методы оптимизации, использующие производные, такие как методы градиентного спуска или Ньютона и их вариации. Полученная точка минимума затем используется как следующая точка для оптимизации целевой функции. Чтобы справиться с возможной невыпуклостью целевой функции и ее аппроксимации, на каждом шаге добавляются случайные нормальные помехи к найденным точкам экстремума. Изменение дисперсии шума позволяет находить глобальный минимум или уточнять локальный.

5.1.1 Аппроксимация целевой функции

Алгоритм DONE строит аппроксимацию $\hat{f}_k$ с помощью случайного ряда Фурье, используя m членов:

$$\hat{f}_k(x) = \sum_{i=1}^{m} w_i cos(\omega_i x + b_i),$$

где вектора частот $\omega_i \in \mathbb{R}^N$ и значения фаз $b_i \in \mathbb{R}$ имеют случайные значения:

$$\omega_i \sim Norm(0_N, \sigma_\omega^2), b_i \sim Unif(0, 2\pi).$$

Вектор амплитуд вычисляется в зависимости от выбранных точек на предыдущих итерациях:

$$Y = [y_1 \dots y_k] \in \mathbb{R}^{N \times K}, t = (t_1 \dots t_k)^T \in \mathbb{R}^K$$

Обозначим $A_k \in \mathbb{R}^{K \times m}$ матрицу значений членов ряда Фурье в точках X:

$$A_k = \begin{pmatrix} cos(\omega_1^T y_1 + b_1) & \cdots & cos(\omega_m^T y_1 + b_m) \\ \vdots & \ddots & \vdots \\ cos(\omega_1^T y_k + b_1) & \cdots & cos(\omega_m^T y_k + b_m) \end{pmatrix}$$

Тогда вектор амплитуд w_k может быть определен как решение задачи минимизации следующего функционала:

$$w_k = \arg\min_w J_k(w)$$

$$J_k(w) = ||y_k - A_k w||^2 + \lambda ||w||^2$$

где λ параметр регуляризации [13].

Так как у функционала J_k существуют производные, то минимум можно найти аналитически, как решение системы линейных уравнений. После всех преобразований получаем:

$$w_i = (A_k^T A_k + \lambda E)^{-1} A_k^T y_k = \Phi_k^{-1} s_k.$$

Полученное выражение можно модифицировать для получения рекуррентного соотношения, что уменьшит вычислительные расходы каждой итерации. Но в этой работе оно не будет рассмотрено.

В результате получаем, задачу оптимизации аппроксимации целевой функции $\hat{f}$ вместо f:

$$y_{k+1} = \arg\min_{y} \hat{f}_k(y), y \in \mathbb{Y}$$

$$\hat{f}_k(y) = \sum_{i=1}^{m} w_i cos(\omega_i y + b_i)$$

5.1.2 Поиск экстремума

Так как функции $\hat{f}_k$ и f могу быть невыпуклы, то алгоритм может сойтись в локальный минимум. Чтобы предотвратить застревание в локальном минимуме, случайные нормально распределенные помехи добавляются к текущей и найденной точке экстремума:

$$y_k = y_k + \zeta_k$$

$$y_{k+1} = y_{k+1} + \xi_k$$

$$\zeta_k \sim Norm(0_n, \sigma_\zeta^2), \xi_k \sim Norm(0_n, \sigma_\xi^2)$$

Дополнительно необходимо обеспечить $y_k, y_{k+1} \in \mathbb{X}$. При выборе больших значений σ_ξ^2 и σ_ζ^2 может быть получена грубая оценка глобального экстремума, при выборе маленьких значений σ_ξ^2 и σ_ζ^2 алгоритм может сойтись в точную оценку локального экстремума. На практике обычно выбирают значения $\sigma_\xi^2 = \sigma_\zeta^2$ [14].

5.1.3 Сравнение с другими методами

Алгоритм NEWUOA [11] решает задачу оптимизации аналогичным образом, но использует аппроксимацию с помощью полиномов второго порядка. В отличие от рядов Фурье, для полиномов второго порядка известно аналитическое выражение точек экстремума, что избавляет от необходимости ре-

шать вспомогательную задачу оптимизации функции аппроксимации. Кроме этого, NEWUOA использует набор техник, позволяющих отбрасывать незначимые точки при построении интерполяции.

Метод Байесовской оптимизации основан на применении кригинга [12]. Используя информацию о доступных наблюдениях целевой функции, строится приближение Гауссовского процесса, среднее значение которого можно считать аппроксимацией целевой функции. С каждой итерацией метода, при добавлении новых точек, уменьшается дисперсия этого процесса. Выбор точек из отрезков с наибольшей дисперсией позволяет находить глобальный минимум, в то время как из отрезков с наименьшей – уточнять локальный.

В отличие от метода NEWUOA, ряды Фурье в алгоритме DONE позволяют аппроксимировать любую непрерывную функцию, в то время как полиномы второго порядка могут аппроксимировать либо выпуклые функции, либо невыпуклые локально. Аналогично методу Байесовской оптимизации, DONE позволяет находить глобальный минимум и уточнять локальный, но, с другой стороны, вычислительная сложность одной итерации алгоритма DONE не зависит от количества уже известных значений целевой функции. По этим причинам для решения задачи определения оптимальной конфигурации кластера был выбран за основу алгоритм DONE.

5.2 Применение для задачи определения конфигурации кластера

Ниже представлен псевдокод алгоритма определения оптимальной конфигурации кластера для вычисления задачи с параметрами x. Он использует информацию о предыдущих K запусках задач X, Y, T: параметры задач, конфигурации кластеров и время выполнения; строит по ним модель зависимости времени работы задачи от конфигурации кластера и предлагает конфигурацию y' для следующего запуска задачи.

Для построения аппроксимации зависимости времени выполнения задач от конфигурации кластера необходимо сначала выбрать набор данных. Для этого информация о предыдущих запусках сортируется по возрастанию меры расстояния $\rho(x, x_i)$ параметров задачи от параметров текущей задачи. В выборку данных для построения аппроксимации попадают все запуски с нулевым расстоянием, и оставшиеся запуски по мере возрастания расстояния, если их конфигурация кластера не конфликтует с уже имеющимися данными в выборке. Размер выборки и максимальное расстояние ограничены сверху константами S_{max} и P_{max} соответственно. В качестве меры расстояния $\rho(\cdot, \cdot)$ было выбрано Евклидово расстояние.

После выбора данных о предыдущих запусках, происходит построение аппроксимации зависимости времени выполнения задачи от конфигурации кластера. Для этого, в соответствии с алгоритмом DONE, случайно выбирается матрица частот и вектор фаз. Вектор амплитуд вычисляется решением системы линейных уравнений $\Phi_k w_k = s_k$. С полученными значениями составляется ряд Фурье, который и является аппроксимацией функции $t = t(x, y)$.

Для поиска точки минимума аппроксимации целевой функции, ее значения вычисляются на всей области значений конфигураций кластера $\mathbb{Y}$. При этом, к текущей и найденным точкам добавляются случайные нормальные помехи, чтобы избежать ситуации застревания в локальном минимуме. Дисперсия помех уменьшается с ростом количества точек, используемых для построения аппроксимации. Найденная точка минимума y' и считается приближением оптимальной конфигурации y^*.

Data: K – количество данных предыдущих запусков

$X = [x_1, \ldots, x_K]; x_i \in \mathbb{R}^M$ – параметры задач

$Y = [y_1, \ldots, y_K]; y_i \in \mathbb{Y} \subset \mathbb{N}^N$ – конфигурации кластеров

$T = (t_1, \ldots, t_K)^T \in \mathbb{R}^K$ – время выполнения

Input: $x \in \mathbb{R}^M$ – параметры текущей задачи

Output: y' – Конфигурация кластера для выполнения задачи с параметрами x

$d \leftarrow \{\rho(x, x_k) : 1 \leq k \leq K\}$

$I \leftarrow \{k : d_k = 0, 1 \leq k \leq K\}$

$y_{in} \leftarrow \{y_k : k \in I\}$

$t_{in} \leftarrow \{t_k : k \in I\}$

if $|y_{in}| < S_{max}$ **then**

$y\_sorted \leftarrow$ отсортировать $\{y_k : 0 < d_k < P_{max}\}$ по возрастанию d_k

$t\_sorted \leftarrow$ отсортировать $\{t_k : 0 < d_k < P_{max}\}$ по возрастанию d_k

$I \leftarrow \{j : 0 \leq j < |y\_sorted|, y\_sorted_j \notin y_{in}\}$

$y_{in} \leftarrow y_{in} \cup \{y\_sorted_k : k \in I\}$

$t_{in} \leftarrow t_{in} \cup \{t\_sorted_k : k \in I\}$

$y_{in} \leftarrow \{y_{in_j} : 0 \leq j < min\{|y_{in}|, S_{max} - |y_{in}|\}\}$

$t_{in} \leftarrow \{t_{in_j} : 0 \leq j < min\{|t_{in}|, S_{max} - |t_{in}|\}\}$

end

$\Omega \leftarrow [\omega_1| \cdots |\omega_m], \omega_i \sim Norm(0_n, \sigma_\omega^2)$

$b \leftarrow (b_1, \ldots, b_m)^T, b_i \sim Unif(0, 2\pi)$

$A_k \leftarrow (cos(\Omega^T y_{in_k} + b))^T, 0 \leq k \leq K$

$\Phi_k \leftarrow A_k^T A_k + \lambda E_m$

$w_k \leftarrow \Phi_k^{-1}(A_k^T y_{in})$

$\hat{f}(x) \leftarrow w_k^T cos(\Omega x + b)$

$y' \leftarrow \arg\min\{\hat{f}(y + \zeta_k), y \in \mathbb{Y}\} + \xi_k$

return y'

Algorithm 1: Алгоритм определения оптимальной конфигурации кластера

Предложенный алгоритм в программной системе предполагается использовать для предоставления пользователям подсказок и значений по умолчанию при выборе конфигурации. Таким образом, пользователь всегда может поменять выбранную алгоритмом конфигурацию на свою.

6 Архитектура программной системы

6.1 Компоненты архитектуры

На рисунке 1 показана высокоуровневая архитектура программной системы. Она состоит из трех главных компонент: веб-интерфейс, управляющий сервис и вычислительная среда. Веб-интерфейс используется конечными пользователями для создания, отправки вычислительных задач и мониторинга их статуса. Веб-интерфейс взаимодействует только с главным управляющим сервисом, обозначенным на рисунке как Idle-Utilizer. Этот сервис ответственен за настройку вычислительной среды, подготовку, выполнение и мониторинг задач пользователя, выгрузку их результатов в облачное хранилище пользователя и анализ истории выполнения задач для оптимизации конфигурации кластера для последующих запусков. Третья компонента архитектуры это вычислительная среда. В случае, если пользователь предпочтет выполнять задачу в облаке, программная система взаимодействует с облачным провайдером для создания виртуального кластера, или, если задачу предпочтительней выполнять на физическом кластере, то файл задачи с входными данными будут подготовлены и отправлены на выполнение в соответствующий планировщик задач. Схема взаимодействия компонент программной системы и пользователя представлена на рис. 2.

Описание веб-интерфейса и сценариев поведения пользователя описаны в [15], а в этой работе опускаются, ввиду несоответствия теме. Для программной системы стоит отметить, только что все взаимодействие веб-интерфейса с главным сервисом происходит через протокол, определенный

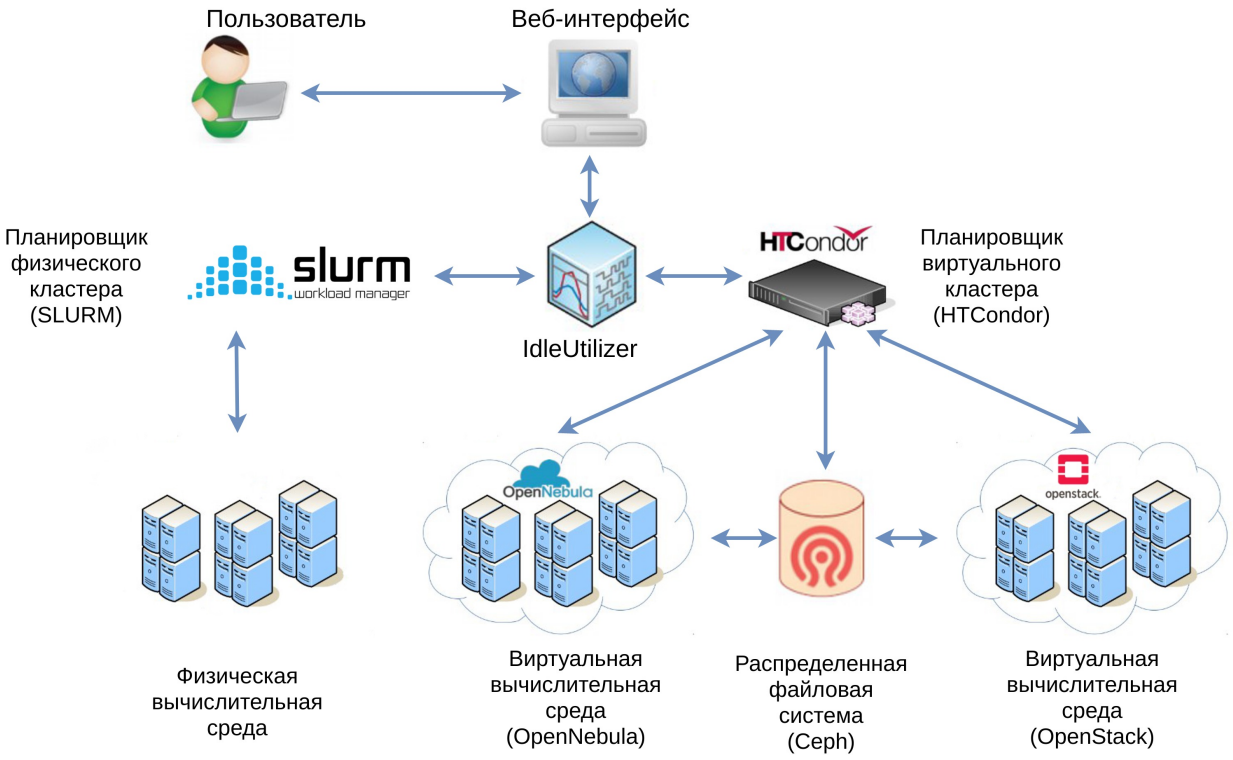

Рис. 1: Высокоуровневая архитектура программной системы.

поверх XML-RPC (remote procedure call). Среди поддерживаемых процедур в основном процедуры для отправки запроса на выполнение и удаление задачи, получение ее статуса и запросы на получение оптимальной конфигурации кластера. При отправке, запрос содержит такие параметры как конфигурация кластера (количество узлов, потоков и памяти), шаблон файла задачи, входные параметры задачи, так же указывается в какой вычислительной среде выполнять вычисления и куда и как выгружать результаты работы задачи. По мере выполнения запроса, дополнительная информация, такая как, например, состояние виртуального кластера, планировщика и статус задачи сохраняются в полях запроса и доступны через RPC-вызовы.

Управляющий сервис (Idle-Utilizer) это главная компонента программной системы. Она ответственна за обработку RPC запросов, управление виртуальной или физической вычислительной средой, отправку и мониторинг вычислительных задач, выгрузку их результатов и выполнение алгоритмов определения оптимальной конфигурации кластера и т.д.. Этот

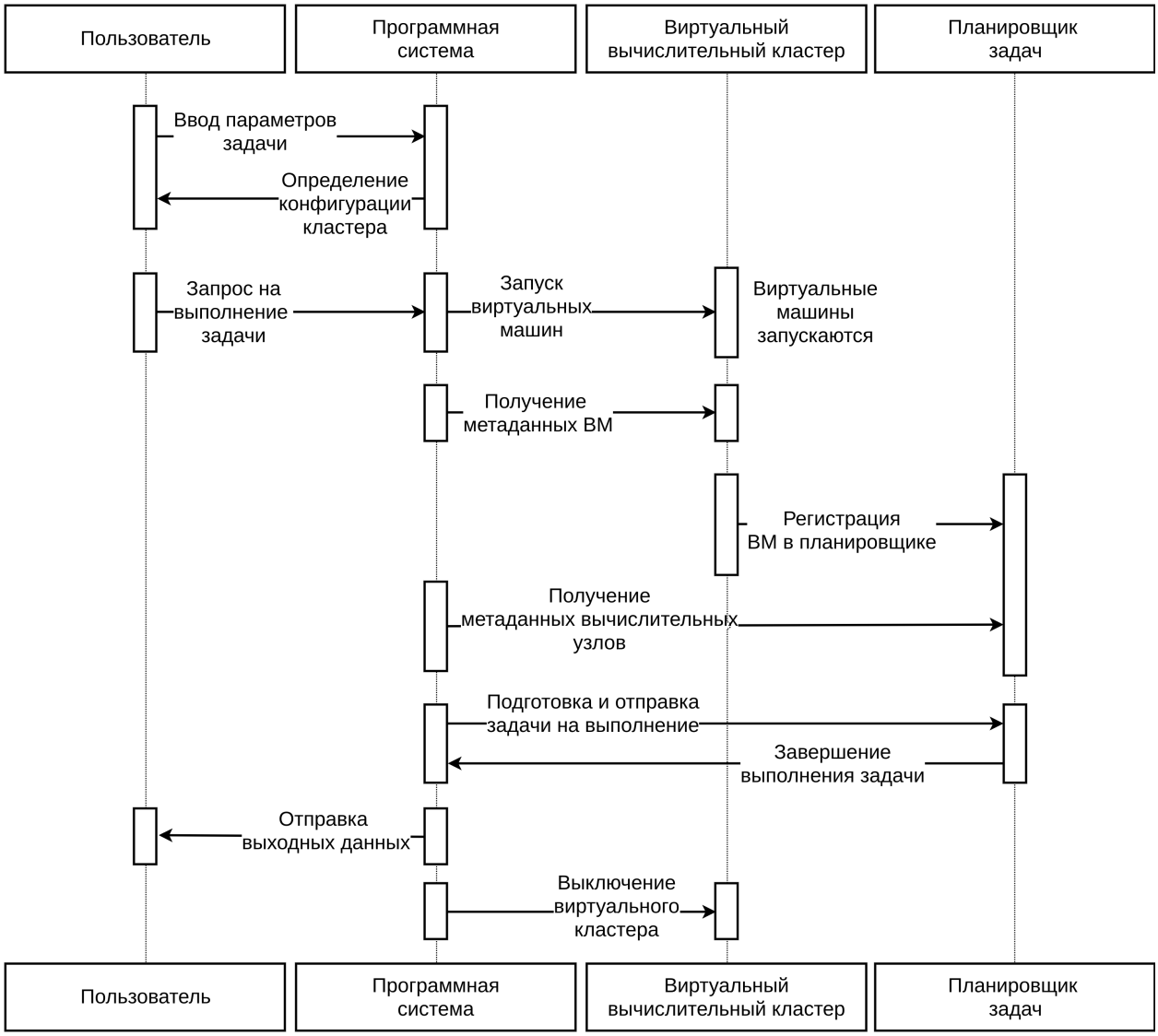

Рис. 2: Взаимодействие компонент программной системы и пользователя.

сервис использует свою базу данных для хранения активных и уже завершившихся запросов. С фиксированным интервалом (2 секунды) этот сервис обновляет состояние каждого запроса, как показано на рисунке 3. В случае возникновения ошибки на любом из этапов обработки запроса, сообщение об ошибке сохраняется в базе данных и сообщается пользователю. Перед и после выполнения каждого из этапов обработки запроса и в случае ошибке, произвольные пользовательские скрипты могу быть запущены, например для тестирования производительности, создания специфичных для задачи файлов или для оправки сообщений об ошибке.

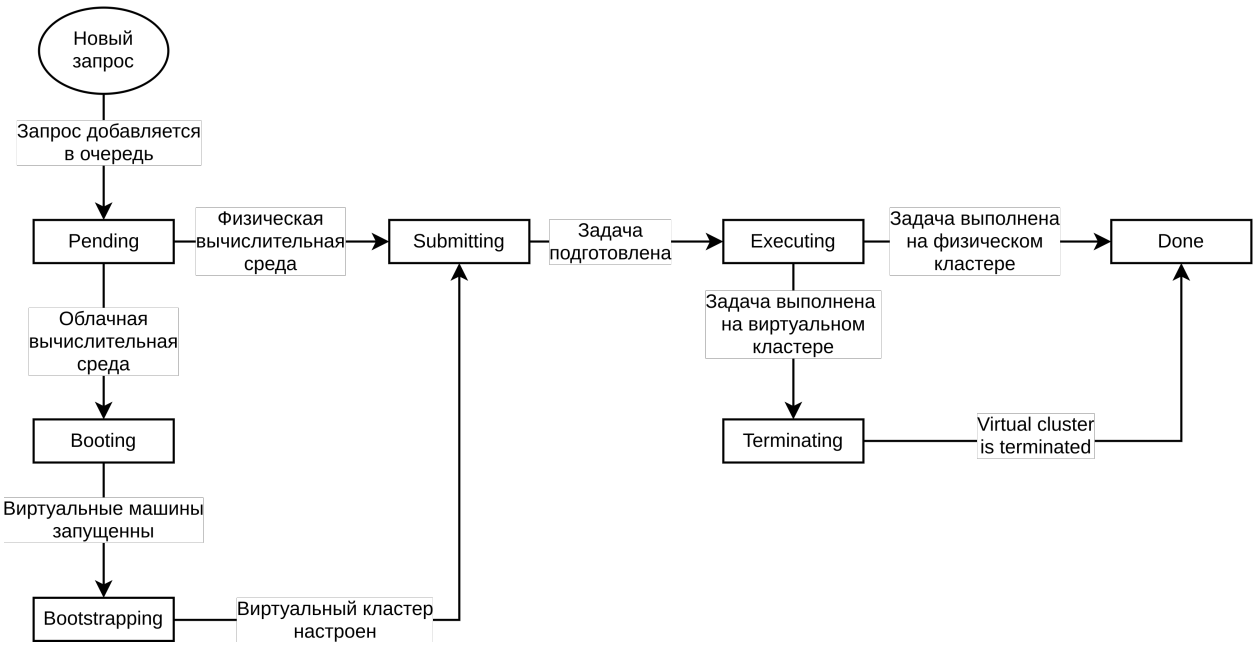

Рис. 3: Изменение состояний запроса при его обработке

6.2 Последовательности обработки запроса

После того, как запрос поступил на выполнение, его состояния начинают изменяться с “pending” до “done” как показано на рис. 3. В состоянии “pending” облачному провайдеру отправляется запрос на создание виртуальных машин, или в случае физического вычислительного кластера, запрос изменяет свое состояние в “submitting”. После того, как виртуальные машины запущены (состояние “booting”), на них запускается программное обеспечение планировщика задач HTCondor и они автоматически регистрируются в пуле узлов планировщика (состояние “bootstrapping”). Во время обработки состояния “submitting”, создается файл задач и отправляется на выполнение в планировщик и запрос переходит в состояние “exectuing”. Когда задача завершена, виртуальный кластер выключается и запрос считается выполненным (состояние “done”).

Пример запроса на выполнения задачи изображен на рисунке 4. Запрос содержит требуемые для выполнения задачи ресурсы (количество узлов, памяти, потоков и процессорного времени) в поле “resources”. Параметры вычислительной среды описаны в поле “backend”. По мере обработки запро-

```
{
        'status': 'submitting',
        'status_updated_at': 1512462493,
        'resources': {
                'nodes': 2, 'memory': 4096, 'cpu': 4, 'time': 100},
        'scheduler': {
                'name': 'htcondor',
                'hosts': [ 'host-100', 'host-101' ]},
        'backend': {
                'name': 'opennebula',
                'owner': {'gid': 1, 'uid': 196},
                'hosts': [
                        { 'id': 100, 'ip': '10.10.0.100' },
                        { 'id': 101, 'ip': '10.10.0.101' }]},
        'template': '....',
        'user_data': {
                'key1': 'value1',
                # Job parametres
        },
        'hooks': [{
            'stage': 'submitting',
            'condition': 'before',
            'name': 'download-input-data.sh ...'
        }]
}
```

Рис. 4: Состояние запроса перед созданием вычислительной задачи и отправки ее в планировщик.

са в это поле добавляется информация об адресах и идентификаторов используемых виртуальных машин. В поле “scheduler” содержит информацию о планировщике задач, используемого в вычислительной среде. В качестве вычислительной среды используется виртуальный кластер, то в это поле будут добавлены имена хостов виртуальных машин. В запросе дополнительно передается шаблон задачи, который состоит из содержимого файла задачи с определенными переменными. Эти переменные конкретизируются значениеми и по шаблону создается файл задачи на этапе “submitting”. Поле “user_data” содержит входные данные задачи и другие параметры, к которым можно получить доступ в шаблоне задачи и в пользовательских скриптах. В поле “hooks” содержится список скриптов, которые будут выполнены на указанном этапе обработки запроса и при определенных условиях.

6.3 Особенности реализации

В текущей версии поддерживается только облачный провайдер OpenNebula и два планировщика задач: HTCondor и Slurm. Если облачный провайдер в запросе не указан, то предполагается, что используется физический вычислительный кластер. В этом случае, работа программной системы сводится к предоставлению интерфейса для работы с планировщиком.

В случае использования виртуальной вычислительной среды, для выполнения каждой задачи создается новый кластер из виртуальных машин. Каждый узел этого кластера создан из одного и того же шаблона и образа. Единственное отличие может быть только в количестве памяти и количества потоков. На узлах виртуального кластера используется планировщик HTCondor и оно настроено автоматически регистрировать себя в пуле узлов планировщика. HTCondor используется только для назначения задач на выполнение узлам виртуального кластера. Хотя такая архитектура может быть упрощена, если отказаться от планировщика и исполь-

зовать только возможности MPI для распределенного выполнения задач, он был оставлен ввиду простоты управления вычислительными задачами и для согласованности работы с физическими вычислительными средами, в которых используются планировщики задач.

Аналогично физическому кластеру, на узлах виртуального кластера смонтирована общая файловая система для запуска распределенных задач и обмена их входными и выходными данными.

Перед выполнением пользовательской задачи, программная система может выполнять произвольные пользовательские скрипты, например, для получения входных данных задачи или генерации их из параметров задачи. После завершения задачи, с помощью аналогичных скриптов результат вычислений может быть выгружен в облачную директорию пользователя.

7 Эксперименты

7.1 Эксперименты с программной системой

Целью проведения экспериментов с программной системой было оправдать использование алгоритмов определения оптимальной конфигурации кластера. Для этого необходимо было показать, что во-первых, программы могут иметь пиковую производительность при конфигурации кластера, отличной от максимальной. В этом случае, если пользователь, будет указывать максимальную конфигурацию кластера, то он может получить замедление времени работы программы. И во-вторых, использование максимальной конфигурации кластера не оправдывает прирост производительности. В этом случае, к примеру, при экспоненциальном увеличении количества узлов кластера пользователь может получить линейный прирост производительности.

Первая ситуация обычно возникает в распределенных приложениях при использовании сетей с низкой пропускной способностью и в параллельных

приложениях при частых синхронизациях параллельного доступа к памяти. Приложения с такими свойствами обычно называют плохо масштабируемыми, и к ним относятся сильно связанные задачи, активно использующие межпроцессорное и межпотоковое взаимодействие. Вторая ситуация возникает почти во всех приложениях согласно закону Амдала, согласно которому ускорение за счет увеличения вычислительных устройств имеет асимптоты, зависящие от доли последовательной части.

В отличие от упомянутых ранее работ, описывающих системы управления виртуальным кластером, эксперименты, показывающие накладные расходы использования виртуальных машин для задач высокопроизводительных вычислений не рассматривались. Их результаты можно найти во многих работах, проведенных разными авторами, например [4, 16].

Для проведения экспериментов был выбран NAS Parallel Benchmarks (NPB), который состоит из нескольких часто используемых программ, спроектированных для тестирования высокопроизводительных систем. Эти тесты состоят из пяти элементарных программ (kernels) и трех приложений, который имитируют задачи, часто возникающие при научных расчетах. Размеры задач в NPB предопределены и обозначаются различными классами. В тестах были использованы задачи MG, FT, CG и IS классов A, B и C:

- IS (Integer Sort): сортировка целых числ, используя метод карманной сортировки.
- CG (Conjugate Gradient): вычисляет метод сопряженных градиентов.
- MG (Multi-Grid): решение уравнения Пуассона используя многосеточный метод.
- FT (Fourier Transform): решение трехмерного уравнения в частных производных, используя быстрое преобразование Фурье.

Производительность тестов CG, MG и FT зависит от пропускной способности сети кластера, тесты IS, CG и MG зависят от скорости работы с памятью. Все перечисленные тестовые приложения распределенные, но не используют параллельные потоки. Количество используемых ими узлов должно быть кратно степени двойки.

Для тестирования программной системы использовались ресурсы облака Объединенного Института Ядерных Исследований [17] под управлением OpenNebula и гипервизером KVM. Так как физические узлы облака были соединены сетью 10 Гбит/с Ethernet, на которой органичней масштабируемости тестов NPB не достигнуть, то в виртуальных машинах пропускная способность сети искусственно уменьшалась, что дополнительно позволило смоделировать работу тестовых приложений на различном оборудовании.

7.2 Эксперименты с алгоритмом

Так как целью работы было спроектировать метод определения конфигурации кластера, а не предоставление версию, готовую для выполнения реальных задач, от экспериментов с ними требуется лишь качественно показать сходимость и правильность работы.

Для тестирования предложенного метода не подходили известные автору тестовые приложения, так как они либо не имели входных параметров, от которых зависел их результат, либо поддерживали слишком мало возможных конфигураций кластеров, либо выполнялись слишком долго. В частности, описанные тесты из NAS Parallel Benchmark представляли из себя распределенные однопоточные приложения, которые требовали количество узлов равное степени двойки и количество их входных параметров ограничивалось только размером класса (A, B или C).

По этой причине были созданы два тестовых приложения с синтетическими нагрузками с использованием технологий MPI и OpenMP. Оба приложения имитируют параллельное выполнение вычислительных подзадач

в распределенном приложении. В первом приложении подзадачи распределяются динамически главным процессом по запросам рабочих процессов. Задачи объединяются и передаются группами фиксированного размера; когда рабочий процесс выполняет все задачи из группы, то он запрашивает новую. Во втором приложении – подзадачи распределяются статически поровну между всеми процессами. Подзадачи состоят из выделения памяти и случайного доступа к ней фиксированное количество раз.

- *Задача 1.* Динамическое распределение подзадач. Параметры: x_1 – размер группы подзадач; x_2 – размер подзадачи
- *Задача 2.* Статическое распределение подзадач. Параметры: x_1 – количество подзадач; x_2 – размер подзадачи

Такие методы распределения задач часто встречаются в вычислительных задачах. В частности, второе приложение моделирует приложения, работающие по принципу “разделяй и властвуй”, к ним относятся различные сеточные методы. Первое приложение моделирует слабосвязанные задачи, например задачи с перебором или задачи на графах.

Для оценки алгоритма было предложено выполнение следующего подхода:

1. Для тестового приложения сохранялось время работы при различных входных параметрах и всех возможных конфигурациях кластера
2. Случайно выбиралась входные данные, конфигурация кластера и соответствующее им время работы. Эти значения использовались как начальное приближение алгоритма и соответствовали самому первому запуску программы.
3. Далее в цикле случайно выбирались входные параметры задачи и они вместе с историей предыдущих запусков подавались на вход алгоритму.

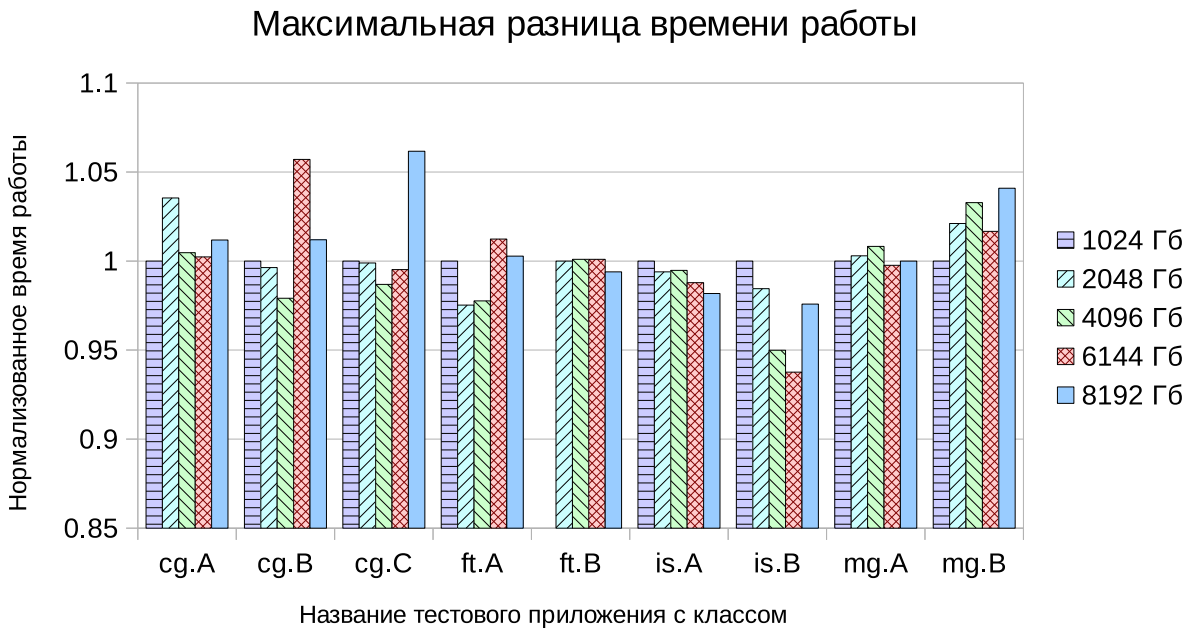

Рис. 5: Максимальное нормализованное время выполнения тестов при разном количестве памяти. Время выполнения при 1024 Гб считается единицей.

4. После каждого такого вызова, алгоритм возвращал конфигурацию кластера, для которой находилось соответствующее время выполнения задачи.

5. Эти значения сохранялись в историю запусков и использовались для последующих запусков алгоритма.

7.3 Результаты экспериментов с программной системой

Тесты были выполнены на виртуальных машинах с разным количеством оперативной памяти. Результаты на рис. 5 показывают, что для каждого тестового приложения может быть выбрано минимальное количество памяти без ущерба производительности. Это можно объяснить тем, тестовые приложения NPB используют фиксированное количество памяти, а не всю доступную.

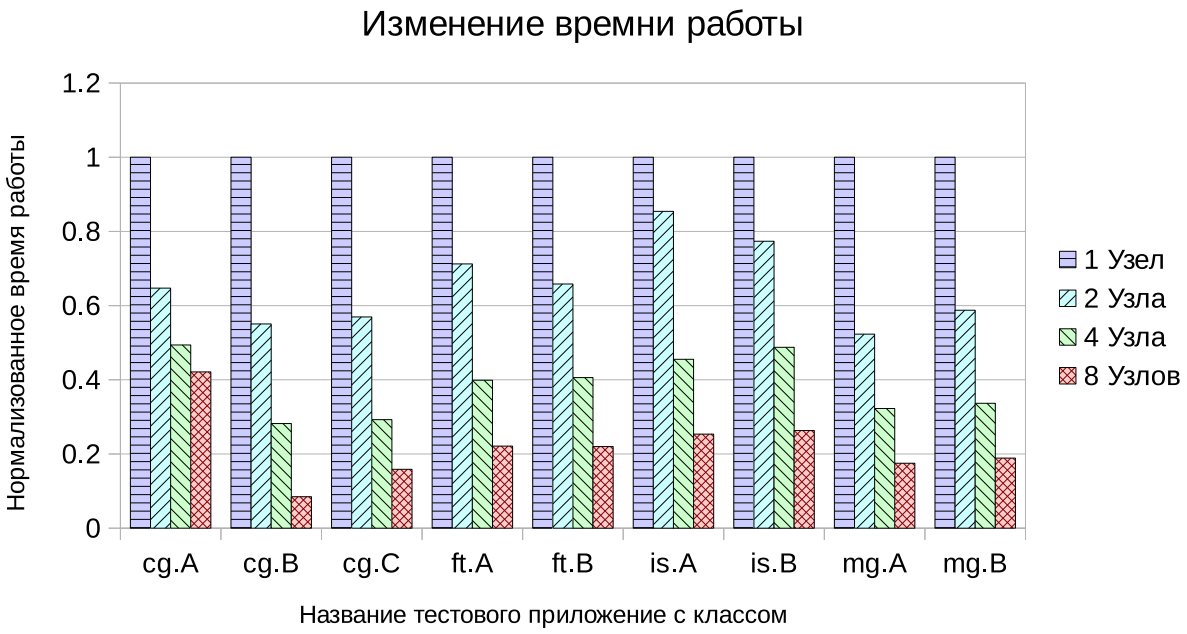

Рис. 6: Изменение времени работы тестовых приложений при увеличении количества узлов.

Тестовые приложения были запущены с разным количеством узлов в виртуальном кластере. На рис. 6 можно заметить, что при увеличении количества виртуальных машин с 4 до 8 в некоторых случаях (особенно на тестах CG и MG) прирост производительности значительно ниже, чем при увеличении их количества с 2 до 4.

На рис. 7 показано выполнения тестовых приложений при различных значениях пропускной способности сети. Эти графики показывают, что при некоторых аппаратных конфигурациях кластера пики производительности могут быть при определенном количестве узлов, а при увеличении количества узлов, время выполнения тестовых приложений увеличивается значительно. Это особенно заметно при пропускной способности сети в 50 Мбит/с, когда оптимальное количество узлов равняется 2 или 4. Кроме этого, можно заметить, что увеличение пропускной способности сети с 1300 Мбис/с до 4400 Мбис/с не дает пропорционального прироста производительности. Хотя пропускная способность сети в текущей задаче не является параметром конфигурации кластера, ее можно использовать для

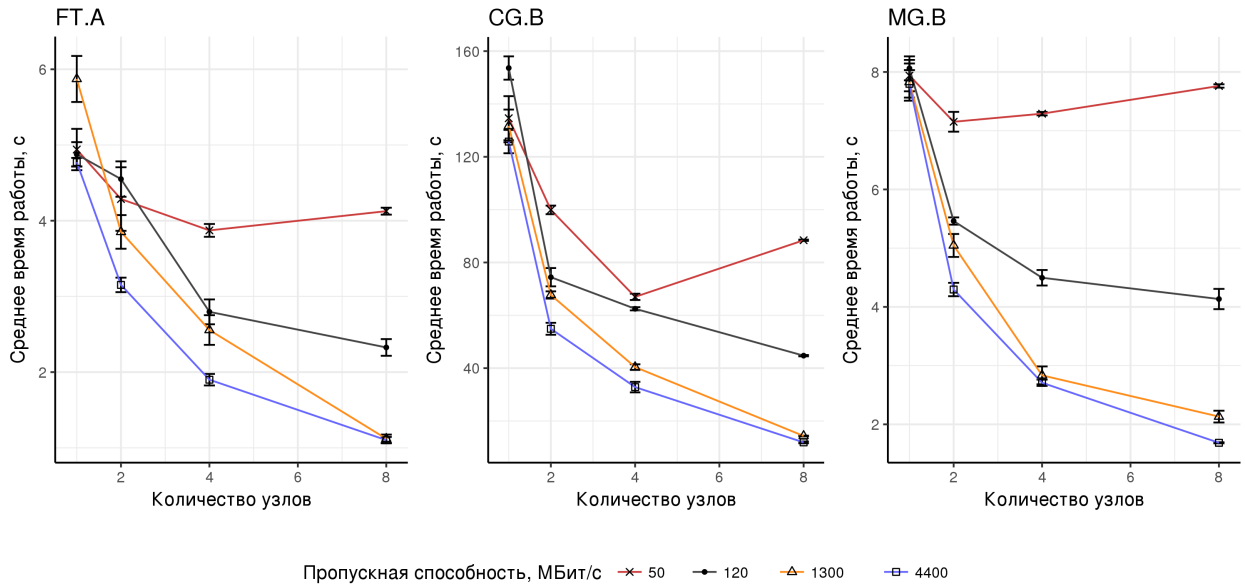

Рис. 7: Время выполнения тестовых приложений при различных значениях пропускной способности сети.

некоторых облачных средах, где ограничивает выбор ресурсов, за которые платят пользователи.

Результаты проведенных экспериментов показывают, что некоторые приложения могут работать оптимальнее при конфигурациях вычислительного кластера, отличной от максимальной. Более того, увеличение конфигурации кластера до максимальной не всегда оправдывает полученный прирост производительности. Так как пользователи разрабатываемой программной системы сами не будут производить такие эксперименты, то имеет смысл автоматизировать определение оптимальной конфигурации кластера.

7.4 Результаты экспериментов с алгоритмом определения конфигурации кластера

Графики зависимости времени работы от конфигурации кластера при разных значениях параметров представлены на рис. 8 и 9. Можно заметить, что первое приложение плохо масштабируется при увеличении количества

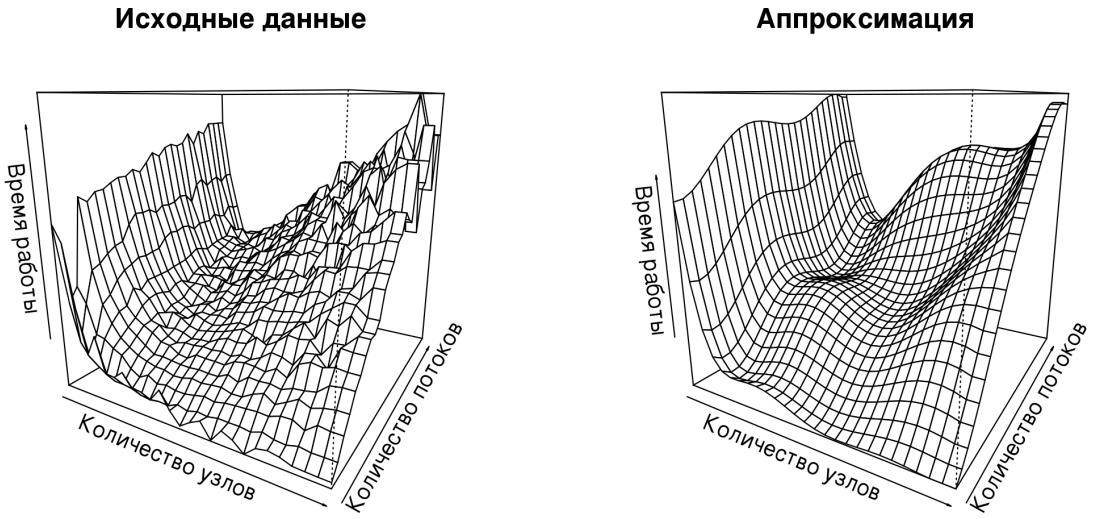

Рис. 8: Аппроксимация зависимости времени работы от конфигурации кластера с помощью ряда Фурье для первого тестового приложения.

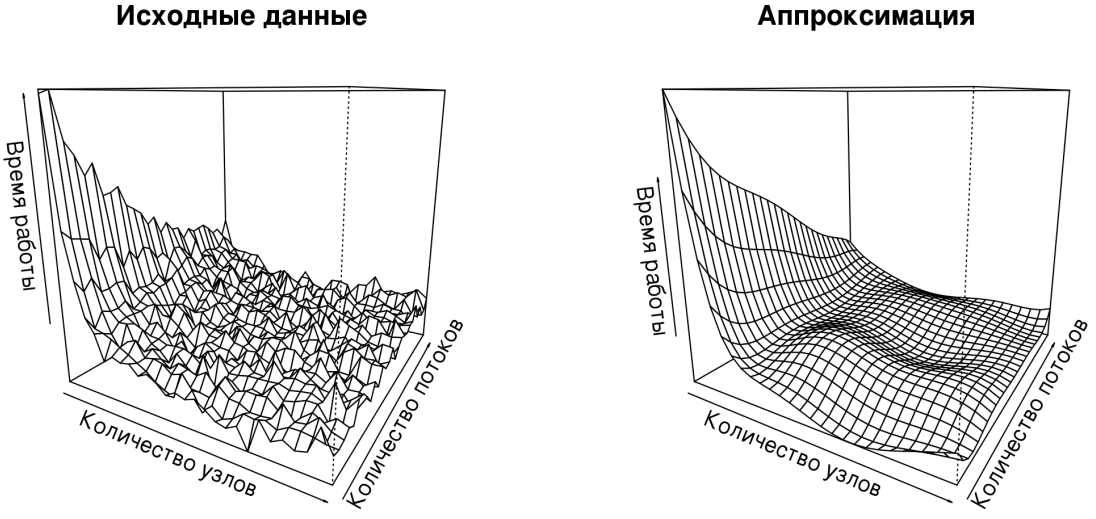

Рис. 9: Аппроксимация зависимости времени работы от конфигурации кластера с помощью ряда Фурье для второго тестового приложения.

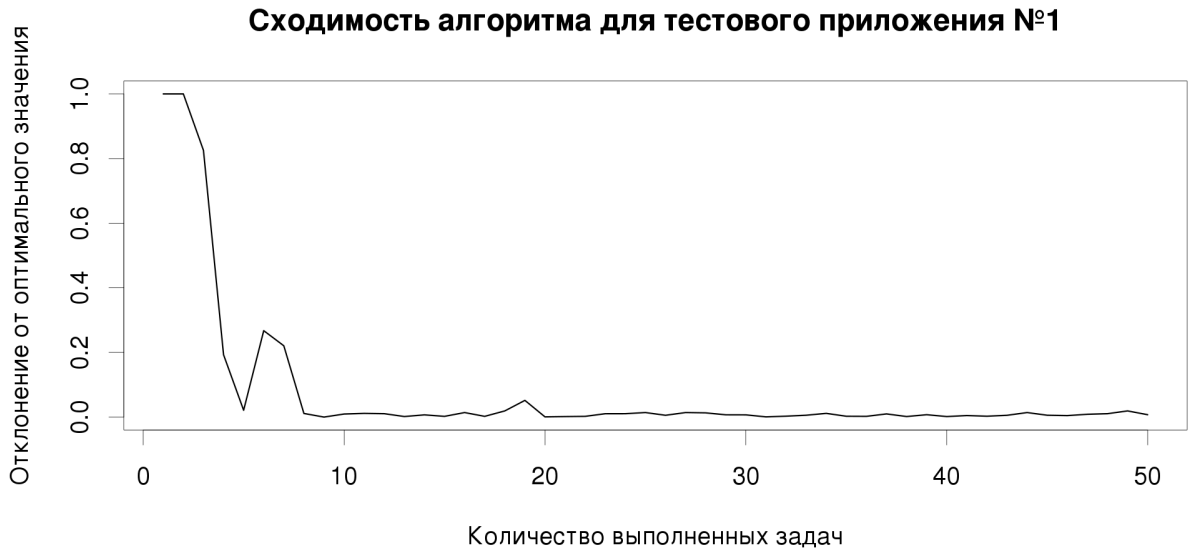

Рис. 10: Отклонение выбранных алгоритмом конфигураций от оптимального значения для первого тестового приложения.

узлов и имеет пики производительности при количестве узлов и потоков отличном от максимального. Справа на графиках представлены аппроксимации этих данных с помощью ряда Фурье, полученные после выполнения алгоритма определения оптимальной конфигурации.

Для оценки отклонения от оптимального значения время выполнения задачи делилось на разницу максимального и минимального времени. Результаты работы представлены на рис 10 и 11. Можно заметить, что алгоритм почти всегда выдает результат, близкий к оптимальному начиная с восьмого запуска задачи.

8 Заключение

8.1 Выводы

В рамках данной работы был спроектирован метод автоматического определения оптимальной конфигурации кластера, минимизирующей время выполнения пользовательской задачи, и была реализована программная си-

Рис. 11: Отклонение выбранных алгоритмом конфигураций от оптимального значения для второго тестового приложения.

стема, предоставляющая пользователям простой интерфейс для выполнения вычислительных задач в облачной среде. Программная система на момент написания работы используется в облачной среде ОИЯИ [17] для выполнения вычислительной задачи моделирования джозефсоновских переходов [18], а спроектированный алгоритм был реализован и протестирован.

8.2 Результаты работы

В рамках данной работы необходимо было решить следующие поставленные задачи:

- Исследовать существующие алгоритмы автоматического определения конфигурации виртуального кластера.

  Алгоритмы были исследованы. Обзор алгоритмов, решающих похожую задачу был приведен в главе “Обзор литературы”. В части “Сравнение с другими методами” главы “Метод определения конфигурации кластера” были описаны аналогичные выбранному методы.

- Спроектировать метод, применимый для решения поставленной цели.

Был спроектирован и реализован метод на основе алгоритма случайного поиска. Описание и псевдокод метода представлены в главе “Метод определения конфигурации кластера”.

- Исследовать существующие решения для запуска HPC приложений в облачной вычислительной среде.
  Существующие решения были рассмотрены их обзор приведен в главе “Обзор литературы”.

- Спроектировать, разработать и протестировать программную систему для запуска приложений в облачной вычислительной среде.
  Программная система была спроектирована, разработана, протестирована. Ее архитектура представлена в главе “Архитектура программной системы”. На момент написания работы, программная система используется в облачной среде Объеденного Института Ядерных Исследований [17] для выполнения задачи моделирования длинных джозефсоновских переходов [18].

- Протестировать приложения при различных конфигурациях кластера, используя программную систему.
  Приложения из NAS Parallel Benchmark были протестированы через программную систему. Результаты экспериментов, приведенные в главе “Результаты эксперементов”, обосновывают необходимость применения алгоритмов автоматической конфигурации виртуального кластера.

- Протестировать сходимость спроектированного метода.
  Метод определения оптимальной конфигурации кластера был реализован и протестирован на двух приложениях с синтетическими рабочими нагрузками. Результаты показывают нахождение оптимальной конфигурации начиная с 8-10 запуска вычислительной задачи.

Тем самым, цель работы “проектирование метода автоматического опре-

деления оптимальной конфигурации кластера и разработка программной системы для запуска вычислительных задач на облачных ресурсах" была достигнута.

Промежуточные результаты работы были опубликованы в двух статьях [15] и [19], выступления теме работы были приняты на конференции ICCSA'18 и CHEP'18.

8.3 Планы на будущее

В качестве дальнейшей работы можно выделить следующее:

- Модификация алгоритма использования методов уменьшения размерности параметров вычислительных задач. В экспериментах рассматривались тестовые приложения с двумя значимыми входными параметрами, но на практике, параметров может быть больше и при этом, не все из них могут влиять на время работы приложения. Поэтому имеет смысл применять методы уменьшения размерности, например факторный анализ, для предварительной обработки данных запуска задач.

- Модификация алгоритма для оптимизации стоимости виртуального кластера ресурсов. Так как программная система разрабатывалась для некоммерческих облачных сред, то первостепенной стояла задача минимизации времени выполнения задач. Но для коммерческих сред, таких как Amazon Web Services, реализованный метод не подойдет, так как его использование может быть слишком затратным. В обзоре других работ приведены методы, минимизирующую стоимость конфигурации кластера, которые можно адаптировать для поставленной задачи.

- Модификация алгоритма для использования данных профилирования работы приложения. Кроме времени выполнения вычислительной задачи, для алгоритма можно использовать данные, собираемые во выполнения задачи. К ним относятся использование памяти, загрузка

центрального процессора, загрузка сети, использование диска и т.д. Полученные данные будут завесить от параметров задачи и конфигурации кластера, и их можно использовать для построения более точной модели регрессии.

- Модификация программной системы для интеграции с другими облачными провайдерами и планировщиками задач. Реализованная программная система имеет достаточно гибкую архитектуру, позволяющую добавлять с одной стороны, типы задач, с другой, вычислительные среды. К примеры, для добавления облачного провайдера достаточно реализовать класс, реализующий низкоуровневые операции по созданию и удалению виртуальных машин.

- Расширение программной системы, чтобы пользователи могли сами регистрировать вычислительные задачи. В текущей архитектуре, только администраторы могут создавать задачи, для этого им нужно, как минимум, скомпилировать и загрузить на сервер исполняемые файлы приложения и настроить веб-форму для указания параметров задачи. Эту задачу можно автоматизировать и делегировать пользователям, что увеличит количество поддерживаемых сервисом вычислительных задач и уменьшит задержки при получении новых научных расчетов.

Список литературы

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[2] Clovis Chapman, Charaka Goonatilake, Wolfgang Emmerich, Matthew Farrellee, Todd Tannenbaum, Miron Livny, Mark Calleja, and Martin Dove. Condor birdbath-web service interface to condor. EPSRC, 2005.

[3] George Ma and Paul Lu. Pbsweb: A web-based interface to the portable batch system. In *12th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS), Las Vegas, Nevada, USA*, 2000.

[4] Frank Doelitzscher, Markus Held, Christoph Reich, and Anthony Sulistio. Viteraas: Virtual cluster as a service. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 652–657. IEEE, 2011.

[5] Wesley Emeneker, Dave Jackson, Joshua Butikofer, and Dan Stanzione. Dynamic virtual clustering with xen and moab. In *International Symposium on Parallel and Distributed Processing and Applications*, pages 440–451. Springer, 2006.

[6] Michael A Murphy, Brandon Kagey, Michael Fenn, and Sebastien Goasguen. Dynamic provisioning of virtual organization clusters. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 364–371. IEEE Computer Society, 2009.

[7] Philip Church, Adam Wong, Michael Brock, and Andrzej Goscinski. Toward exposing and accessing hpc applications in a saas cloud. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 692–699. IEEE, 2012.

[8] Navendu Jain, Ishai Menache, and Ohad Shamir. On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud. 2014.

[9] Fangzhe Chang, Jennifer Ren, and Ramesh Viswanathan. Optimal resource allocation in clouds. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 418–425. IEEE, 2010.

[10] Hans RGW Verstraete, Sander Wahls, Jeroen Kalkman, and Michel Verhaegen. Model-based sensor-less wavefront aberration correction in optical coherence tomography. *Optics letters*, 40(24):5722–5725, 2015.

[11] Michael JD Powell. The NEWUOA software for unconstrained optimization without derivatives. In *Large-scale nonlinear optimization*, pages 255–297. Springer, 2006.

[12] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[13] Ali H Sayed and Thomas Kailath. Recursive least-squares adaptive filters. *The Digital Signal Processing Handbook*, 21(1), 1998.

[14] Laurens Bliek, Hans RGW Verstraete, Michel Verhaegen, and Sander Wahls. Online optimization with costly and noisy measurements using random fourier expansions. *IEEE transactions on neural networks and learning systems*, 2016.

[15] N. A Balashov, M. V. Bashashin, R. I. Kuchumov, N. A. Kutovskiy, and I. A. Sokolov. JINR Cloud Service For Scientific And Engineering Computations. *Accepted at CHEP*, 2018.

[16] Aravind Menon, Jose Renato Santos, Yoshio Turner, G John Janakiraman, and Willy Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 13–23. ACM, 2005.

[17] A. V. Baranov, N. A. Balashov, N. A. Kutovskiy, and R. N. Semenov. JINR cloud infrastructure evolution. *Physics of Particles and Nuclei Letters*, 13(5):672–675, 2016.

[18] M.V. Bashashin, E.V. Zemlyanaya, I.R. Rahmonov, Y.M. Shukrinov, P.K. Atanasova, and A.V. Volokhova. Numerical approach and parallel implementation for computer simulation of stacked long josephson junctions. *Computer research and modeling*, 8(4):593–604, 2016.

[19] R. Kuchumov, V. Petrunin, V. Korkhov, N. Balashov, N. Kutovskiy, and Sokolov I. Design and implementation of a service for cloud HPC computations. *Accepted at International Conference on Computational Science and Its Applications*, 2018.