Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

РАБОТА ПРОВЕРЕНА
Генеральный директор
ООО «Наполеон Айти»
_____ / П.С. Подкорытов
«___» _____2018 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, к.ф-м.н, директор ВШЭКН:
_____ / Г.И. Радченко
«____» _____ 2018 г.

Разработка программного комплекса для организации масштабируемых научных вычислений в гетерогенных средах

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ – 09.04.01.2018.488.ВКР

Руководитель:
_____/ Г.И. Радченко
к.ф-м.н, Директор ВШЭКН ЮУрГУ
«_____» _____ 2018 г.

Автор работы:
_____/ В.В. Кашанский
студент группы КЭ - 222
«_____» _____ 2018 г.

Нормоконтролер:
_____/ _____
_____
«_____» _____ 2018 г.

Челябинск 2018

АННОТАЦИЯ

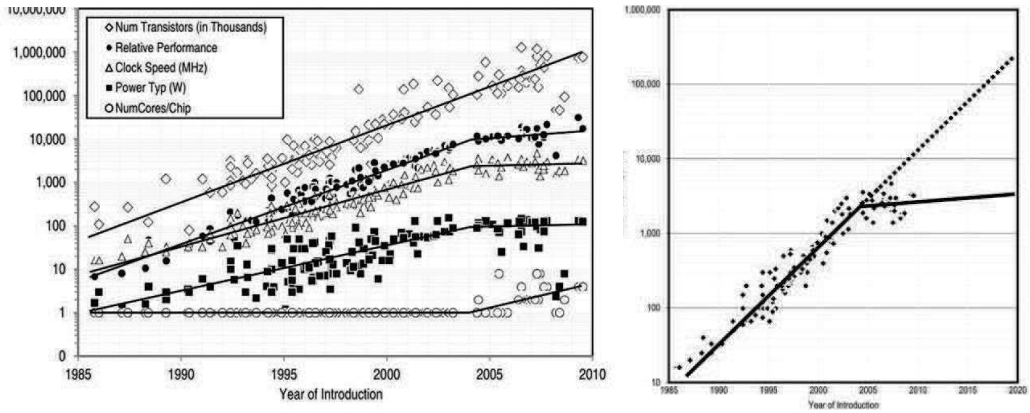
Кашанский В.В. Разработка программного комплекса для организации масштабируемых научных вычислений в гетерогенных средах – Челябинск: ВШЭКН ЮУРГУ, ЭВМ; 2018, 85 с. 23 ил., библиогр. список – 41 наим., 3 прил., 6 листов чертежей ф. А4.

В рамках выпускной квалификационной работы производится детальный анализ современных технологий организации вычислительных сред. Организуется разработка программного комплекса для организации масштабируемых научных вычислений в гетерогенных средах с использованием ПО BOINC, GlusterFS и SLURM. Производится выборка и анализ результатов работы системы, в домене специально разработанных задач. Рассматриваются преимущества и недостатки, как модели вычислений в гетерогенных средах в целом, так и конкретного программного комплекса. Доказывается способность предлагаемой архитектуры к обеспечению успешного цикла решения задач определенного класса.

Оглавление

ВВЕДЕНИЕ

На сегодняшний день сектор исследований, связанный с информационными технологиями демонстрирует интенсивный качественный и количественный рост, претерпевая значительные трансформации. Каждый год предлагаются новые аппаратные архитектуры, программные продукты, публикуются фундаментальные работы различного уровня абстракции. Рост исследовательского интереса не случаен и является закономерным ответом на радикально изменяющиеся запросы к системам обработки информации. В частности, исследования в области биоинформатики, прецизионной медицины и физики высоких энергий генерируют не только огромные массивы данных и вычислительных задач, но и принципиально новые качественные требования к масштабам физических процессов и типам систем-обработчиков.



а) б)

Рисунок 1. а) диаграмма развития современной вычислительной архитектуры (x86 Intel, AMD); б) Динамика тактовых частот вычислительных архитектур с 1985 по 2020 год (точками «предполагаемые», сплошной линией – «реальные»)

В частности, одной из важных проблем последнего десятилетия стало то, что закон вертикального масштабирования, описанный Муром [1] и Деннардом [2] перестал выполняться. Так, на момент 2018 года число транзисторов продолжает увеличиваться, но темпы роста производительности тактируемых процессоров достигли участка насыщения (Рис. 1 а и б). Закон масштабирования упирается в определенные пределы выделяемой мощности процессора, после которых процессоры просто плавятся от перегревания. И преодолеть эти пределы невозможно без использования нетрадиционных, громоздких и дорогих систем охлаждения или радикальных инфраструктурных преобразований. Глядя на рисунок 1, можно заметить, что с 2006 года тактовая частота массовых процессоров не растет выше примерно 4ГГц.

Несоблюдение Закона Деннарда и как следствие невозможность наращивать больше тактовую частоту процессоров привели к тому, что производители обратились к нескольким альтернативам:

- производство многоядерных процессоров;
- производство специализированных вычислителей;
- изменение алгоритмов и вычислительных концепций.

На фоне невозможности дальнейшего энергоэффективного наращивания тактовых частот развилось целое множество моделей распределенных вычислений, и в частности ГРИД.

Напомним, что ГРИД-технологии позволяют создать географически распределенные вычислительные инфраструктуры, которые объединяют разнородные ресурсы и реализуют возможность коллективного доступа к этим ресурсам. Принципиальной новизной этих технологий является объединение ресурсов путем создания компьютерной инфраструктуры нового типа, обеспечивающей глобальную интеграцию информационных и вычислительных ресурсов на основе сетевых технологий и специального программного обеспечения промежуточного уровня (middleware), а также

набора стандартизованных сервисов (служб) для обеспечения надежного совместного доступа к географически распределенным информационным и вычислительным ресурсам: отдельным компьютерам, кластерам, хранилищам информации и сетям. Основными направлениями развития грид-технологий являются: вычислительный грид, грид для интенсивной обработки данных и семантический грид для оперирования данными из различных баз данных. [3] В статьях [4] и [5] дано более формальное описание термина ГРИД и всей взаимосвязанной с ним семантики.

Стоит отметить, что время не стоит на месте,произошли серьезные изменения и сегодня, на фоне активного развития технологий контейнеризации и виртуализации, активную известность приобрела концепция облачных вычислений. Вопрос о том к какой системе понятий отнести разрабатываемую нами систему является спорным и открытым, так как в процесс создания и эксплуатации системы вовлечено сразу несколько уровней абстракции.

Важно одно - не смотря на развитие облачных технологий, крупномасштабных ГРИД систем, важной проблемой остается интеграция подобных систем друг с другом. На сегодняшний день провайдеры облачных предоставляют легко масштабируемые облачные ресурсы, мы так же располагаем собственными ресурсами как ПК и кластеры, ресурсами волонтеров. Более того - миллионами браузеров, поддерживающих тьюринг-полные скриптовые языки программирования. Но как конкретно организовать вычисления - нет единого мнения и решения. В частности один только выбор корректной файловой системы для того или иного эксперимента представляет собой задачу, требующую сложного многокомпонентного анализа. А системы обработки вычислительных заданий выдвигают ряд ещё более серьезных требований [3-5].

Таблица 1 - Динамика емкостей и вычислительной производительности различных устройств за последние 20 лет [6]

	Hard Disk Drives		SSD (MLC Flash)		DRAM		Ethernet
Year	Capacity	Throughput	Capacity	Throughput (r/w)	Capacity	Throughput	Bandwidth
1993					16 Mibit/chip†	267 MiB/s†	
1994	4.3 GB†	9 MB/s†					
1995							100 Mbit/s†
2003	73.4 GB†	86 MB/s†					10 Gbit/s†
2004					512 Mibit/chip	3.2 GiB/s	
2008			160 GB	250/100 MB/s[3]			
2012			800 GB	500/460 MB/s[4]			
2014	6 TB	220 MB/s[1]	2 TB	2.8/1.9 GB/s[5]	8 Gibit/chip	25.6 GiB/s	100 Gbit/s
2015	8 TB	195 MB/s[2]					
Improvement	×1860	×24	×12.5	×11.2	×512	×98	×1000

[1] Seagate ST6000NM0034, http://www.storagereview.com/seagate_enterprise_capacity_6tb_35_sas_hdd_review_v4

[2] Seagate ST8000AS0012 (SMR), http://www.storagereview.com/seagate_archive_hdd_review_8tb

[3] Intel X25-M, http://www.storagereview.com/intel_x25-m_ssd_review

[4] Intel SSD DC S3700, http://www.storagereview.com/intel_ssd_dc_s3700_series_enterprise_ssd_review

[5] Intel SSD DC P3700, http://www.tomshardware.com/reviews/intel-ssd-dc-p3700-nvme,3858.html

Актуальность исследовательской темы, связанной с исследованиями в области масштабируемых научных вычислений в гетерогенных средах заключается в том, что модель распределенных вычислений, в частности, облачные технологии и ГРИД стали **особо важной частью** научного мира, можно даже сказать – ключевой. В течение последних десятилетий произошла интенсивная стандартизация и глобализация использования всех видов компьютерных ресурсов, вызванная **ростом сложности решаемых задач**. Более того, функционирование многих фундаментальных исследовательских проектов стало невозможным в принципе без высокопроизводительных вычислительных систем. Ясно, что фундаментальные и прикладные исследования в области мультимасштабного компьютинга могут привести к качественным изменениям, как в научном мире, так и в области коммерческих решений.

Целью представленной выпускной квалификационной работы является разработка программного комплекса обеспечивающего объединение гетерогенных распределенных ресурсов с использованием ПО BOINC, SLURM и GlusterFS.

Для достижения поставленной цели, необходимо решить следующие поставленные задачи:

1. Рассмотреть существующие научные проекты, генерирующие потоки данных и вычислительных задач с различной степенью интенсивности;
2. Провести детальный анализ современных программных технологий для организации масштабируемых вычислений в контексте гетерогенных вычислительных сред;
3. Разработать архитектуру собственного программного комплекса на базе существующих аппаратных решений и представить конечную её реализацию с использованием ПО BOINC, GlusterFS и SLURM;
4. Оценить работоспособность разработанного программного комплекса в различных режимах и внешних условиях (интенсивность потоков данных и задач, инфраструктурная конфигурация);
5. Произвести выборку результатов и проанализировать работу системы, в домене специально разработанных задач.

ГЛАВА 1. АНАЛИЗ СУЩЕСТВУЮЩИХ НАУЧНЫХ ПРОЕКТОВ

1.1 Информация о проекте Worldwide LHC Computing GRID

На сегодняшний день самой крупной, межгосударственной ГРИД сетью оперирует проект WLCG (Worldwide LHC Computing GRID) [7-9] организованный в ЦЕРНе с целью создания глобальной информационно-вычислительной инфраструктуры (Рис 2.) для обработки, хранения и анализа данных, полученных во время различных экспериментов, проводимых на Большом адронном коллайдере [10-12] Для реализации этой задачи построена глобальная грид-инфраструктура на основе региональных центров различного уровня, обеспечивающая моделирование, хранение и передачу данных с LHC.

Рисунок 2. Карта участников сети уровня 0,1,2 (Tier 1,2) [9]

WLCG спроектирован в CERN и предназначен для обработки больших объёмов данных, поступающих с LHC (Рис. 3) включает в состав 170 вычислительных центров из 36 стран. Грид LCG был запущен 3 октября 2008 года, как распределённая сеть, которую CERN разработал для обработки огромных объёмов данных, получаемых в LHC. Она включает в себя как

частные волоконно-оптические кабели связи, так и существующие высокоскоростные части общественного Интернета [3].

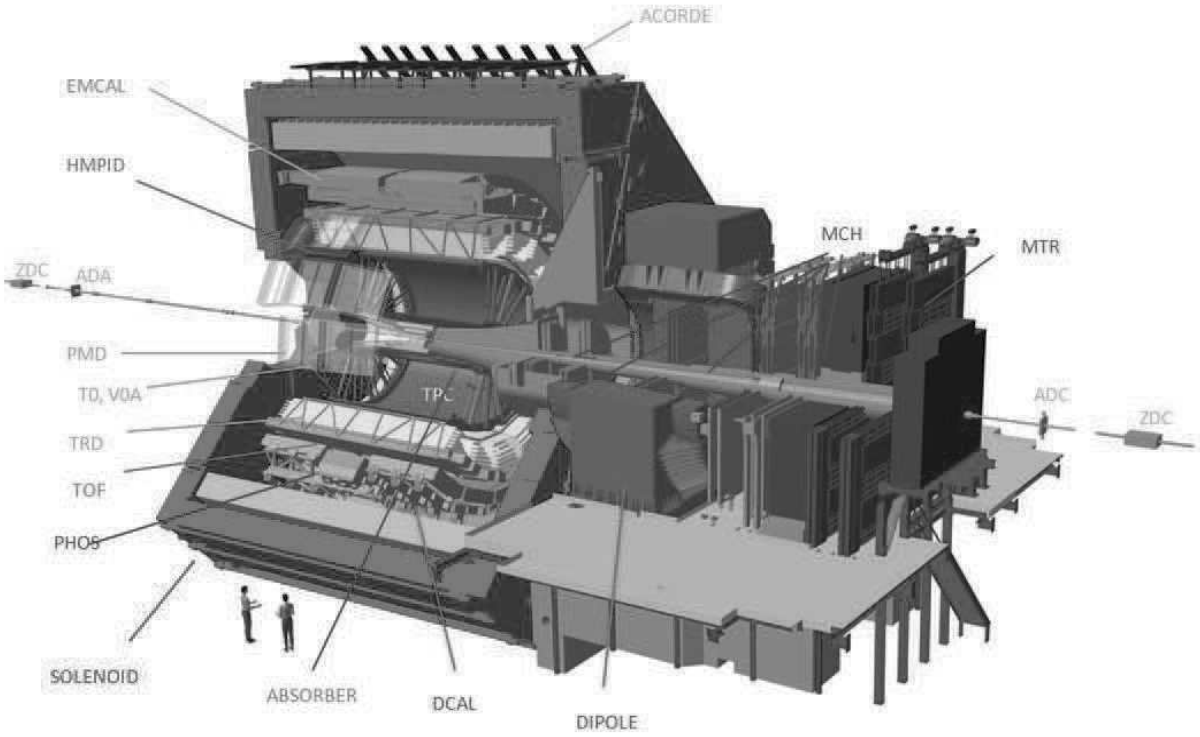


Рисунок 3. Происхождение данных WLCG. Эксперимент ALICE [13].

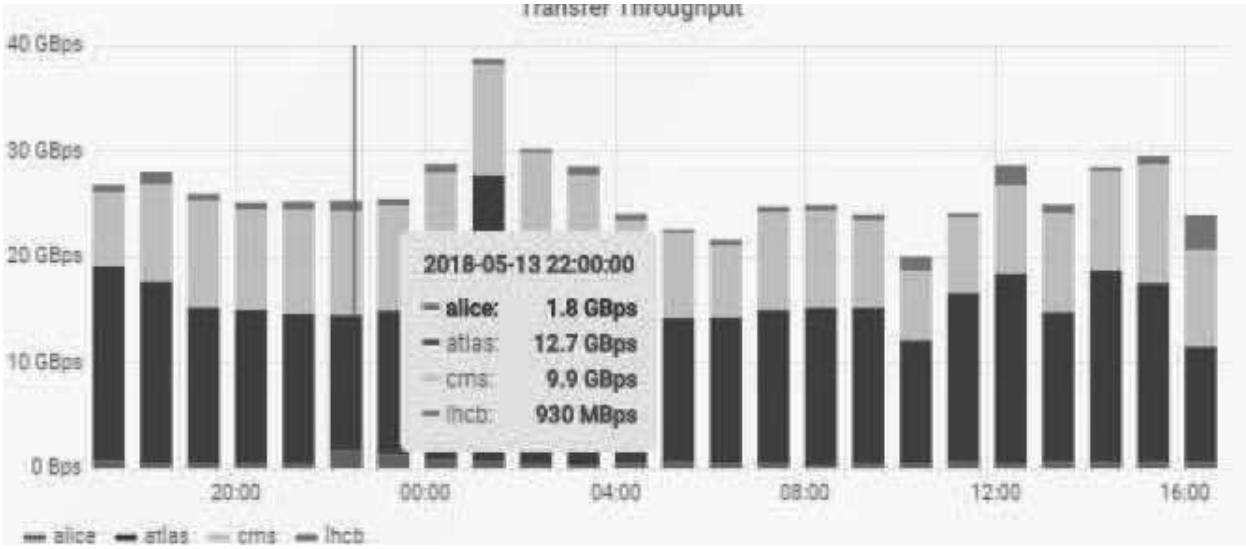


Рисунок 4. Суточные колебания интенсивностей потоков данных для различных экспериментов [9]

На рисунке 4 представлены величины общесистемных интенсивностей потоков данных в сети на момент 13 мая 2018 года, теоретические расчеты и более старые данные указаны в [7-8].

WLCG сеть состоит из четырех уровней, называемых “Tiers” (ярус, уровень). Всего существует 4 уровня, соответственно 0, 1, 2, 3. Каждый уровень организован группой компьютеров и вычислительных центров. При том, с ростом индекса уровня вычислительные возможности предполагаются более слабыми с точки зрения вычислительных емкостей по отношению к предыдущим [7-8].

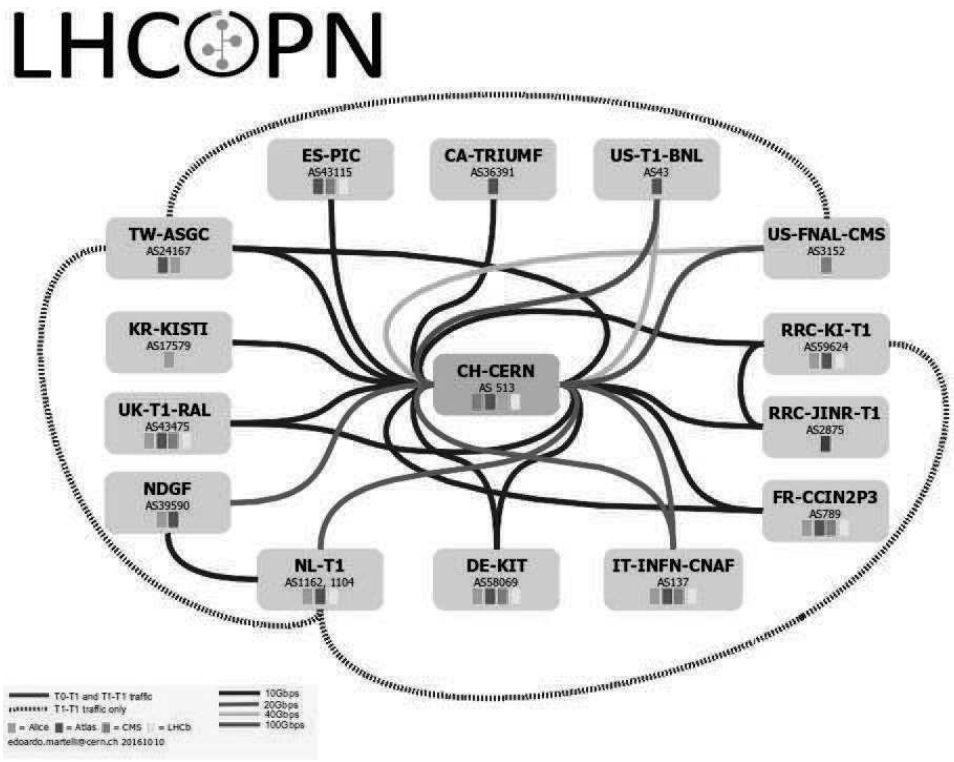


Рисунок 5 Архитектура Worldwide LHC Computing Grid

Вычислительный центр Tier 0 принадлежит CERN и располагается в организации по близости с городом Женева, Швейцария. Данные со всех экспериментов поступают на централизованный сервер. Однако центр не сохраняет все генерируемые данные. Основная задача центра - извлекать экспериментальные данные без ошибок (обеспечить целостность потока) и перераспределить информационные потоки среди центров более низких уровней.

Вычислительные центры Tier 1 (Рис. 5) представлены крупными вычислительными узлами. Такие центры обязаны удовлетворять определенным строгим критериям, таким как: устойчивость, возможность обрабатывать и хранить большие массивы данных, высокая пропускная способность сети - минимум 10 Gbit/s на канал. Как правило, для хранения данных в таких центрах используются ленточные роботы IBM, так же SSD и HDD (SAS, SATA) системы и специализированное ПО – PanDA, Intel Lustre, IBM Spectrum Scale, dCache, GlusterFS и.т.п. Все они получают данные от

вышестоящего ДЦ – Tier 0 и выполняют необходимые манипуляции с данными.

Tier 2 представляют собой центры анализа данных, которые обеспечивают вычислительные мощности для моделирования, необходимого в рамках экспериментов. Результаты этих симуляций отправляются на Tier 1 для обработки и архивирования. Структура и требования к сайтам Tier 2 варьируется от страны к стране. В некоторых случаях, сайт поддерживает только один эксперимент. В Европе обычно поддерживается несколько или даже все эксперименты сразу. Центры уровня 2 предоставляют только вычислительные и дисковые ресурсы, без архивирования данных [7]. Таким образом, центры Tier 2 более просты в обслуживании и часто управляются маленькой командой инженеров.

Узлы Tier 3,4 предполагаются самыми слабыми звеньями сети. Предполагается, что они вообще не способны обеспечить устойчивую работу оборудования, возможность обрабатывать и хранить большие массивы данных, а так же высокую пропускную способность канала. Узлы Tier 3,4 взаимодействуют с центрами Tier 2 и как правило, ресурсный пул находятся в обыкновенных пользовательских сетях, которые к тому же не являются выделенными и могут быть перегружены.

Более подробное описание системы можно найти в работах [7-8], [10-12] а так же на сайте проекта [9].

1.2 Информация о проекте Human Brain Project

Другим известным экспериментом является проект «Human Brain» [14-15]**.**проводимый под руководством Федеральной политехнической школы Лозанны, Швейцария. Платформы HBP состоят из прототипа аппаратного обеспечения, программного обеспечения, баз данных и программных интерфейсов. Эти инструменты доступны для исследователей во всем мире через лабораторию HBP. [16]

Проект развивает исследования в шести областях [17]

- Нейроинформатика
- Моделирование работы мозга
- Высокопроизводительная аналитика и вычислительная техника
- Медицинская информатика
- Нейроморфные вычисления
- Нейророботизация

Особый интерес представляет направление «Высокопроизводительная аналитика и вычислительная техника» [17] , в рамках которого проектируется и реализуется распределенная вычислительная инфраструктура для поддержки проекта Human Brain.

По данным исследователей проекта, в математических моделях нейросетей для представления детальной электрофизиологии и связи одного нейрона на масштабах межклеточного взаимодействия может потребоваться до $2 * 10^4$ дифференциальных уравнений [15-16] Даже современные многоядерные рабочие станции сталкиваются с трудностями при решении подобного числа уравнений для моделирования биологических процессов, протекающих в реальном времени. Другими словами, единственный вариант, который бы обеспечил разумные времена расчетов при моделирования сетей нейронов с заданной на перед точностью[1], - это использование высокопроизводительных вычислений (HPC) и ГРИД технологий.

Модель неокортикальной колонки[2] включает в себя подробные представления о, по меньшей мере, $3 * 10^4$ нейронов [15-16]. В большинстве случаев в это число может значительно варьироваться, чтобы обеспечить

[1] Частота дискретизации оказывает прямое влияние на время решения и требования к вычислительной архитектуре.
[2] Ориг. - Neocortical column

допустимые граничные условия. Моделирование работы головного мозга на том же уровне детализации представляло бы до $200 * 10^6$ нейронов и потребовало бы по представлениям ученых проекта, примерно в 10 000 раз больше памяти. Для имитации человеческого мозга потребуется еще 1000-кратное увеличение объема памяти и вычислительной мощности. Ясно, что дальнейшее масштабирование до субклеточного моделирования, к примеру, нейролепальной сосудистой системы и создание виртуальных инструментов (например, виртуальная ЭЭГ, виртуальная ФМР) дополнительно расширят эти требования.

На начальном этапе своей работы в проекте предшественнике – «Blue Brain» использовался суперкомпьютер IBM BlueGene/L с 8192 ядрами. Позднее использовались 16384 основных ядра IBM BlueGene/P и почти в 8 раз больше памяти, чем у предшественника. Сегодня проект использует суперкомпьютер IBM BlueGene/Q с 65536 ядрами, сопроцессорами и расширенными возможностями памяти, организованными в Швейцарским национальным суперкомпьютерным центре (CSCS) в Лугано. [18]**.** На сегодняшний день проект "Human Brain" объединяет несколько вычислительных центров по всей европе (Рис. 6) в единую платформу - The High Performance Analytics and Computing (HPAC) [17], [19-20].

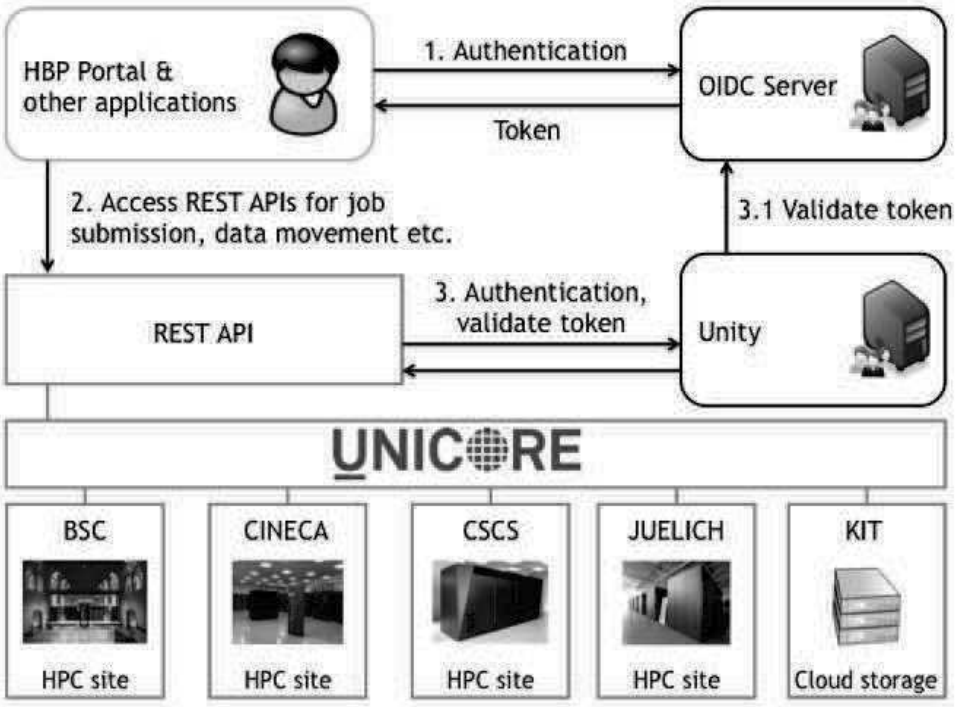


Рисунок 6. Диаграмма взаимодействия между пользователем и вычислительными центрами в рамках единой ГРИД платформы - The High Performance Analytics and Computing (HPAC) [19]

Платформа HPAC (Рис. 6) предоставляет унифицированный интерфейс к суперкомпьютерам, системам хранения и визуализации, позволяя оперировать сразу нескольким исследовательским проектам. В частности, исследовательские группы могут:

- Запускать высокопроизводительное мультимасштабное моделирование на основе разработанных систем уравнений.
- Размещать крупные объемы данных на площадках, входящих в консорциум.
- Оркестрировать потоки задач, путем взаимодействия с Workflow-системами

1.3 Общая схема обработки данных с использованием распределенной вычислительной системы

В предыдущих параграфах мы произвели рассмотрение двух наиболее значимых, всемирно известных мегасайенс эксперимента в области физики высоких энергий и мультимасштабного нейромоделирования. Оба проекта характеризуются большими объемами данных, как в статическом случае, так и в динамическом. Помимо данных, так же генерируются большие объемы вычислительных задач для целого ряда целевых платформ и архитектур. Рассматривая инфраструктуру проектов в деталях можно выявить наиболее важные особенности и закономерности для информационных систем, функционирующих в рамках подобных проектов.

Главным образом, каждый эксперимент характеризуется генератором событий (Рис. 7) – некоторым объемом, который является истоком событий для системы сбора данных (Рис. 8). На практике по мимо системы детекторов, захватывающих «реальные» физические данные абсолютно любой природы, данные могут генерироваться на основе программного обеспечения.

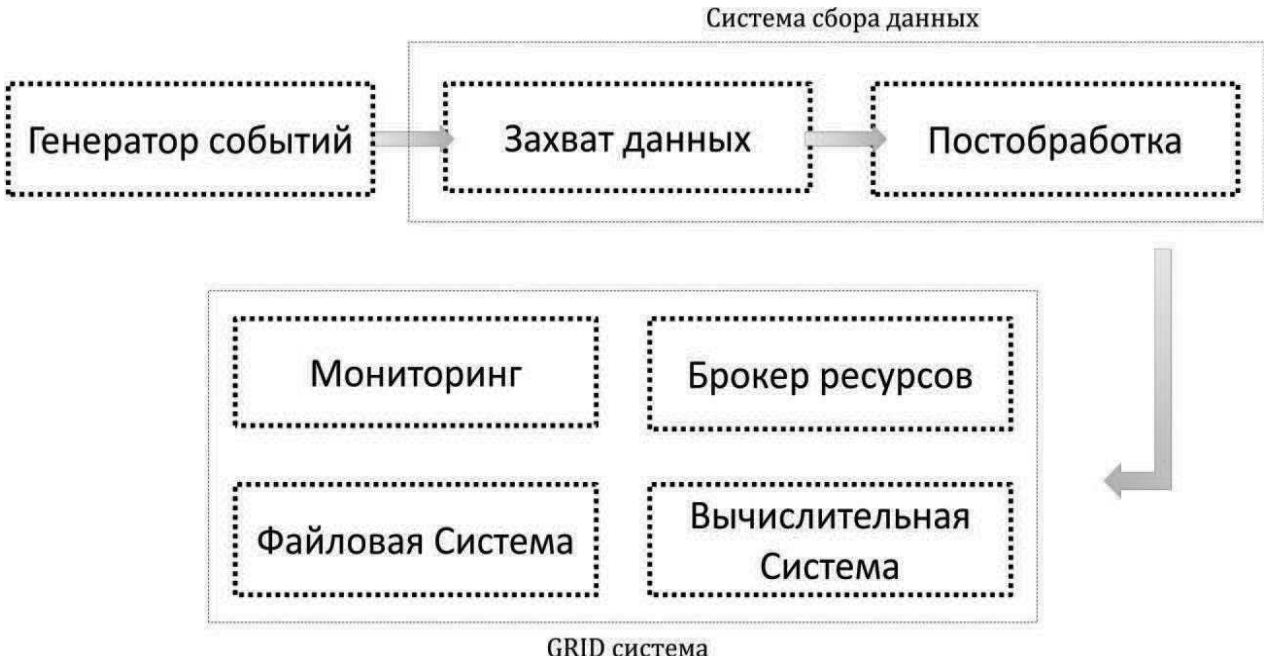


Рисунок 7. Схема взаимодействия ГРИДа и системы сбора данных

Возможны так же комбинации вышеприведенных вариантов. К тому же данные уже могут быть собраны и ожидать обработки. В случае высоких интенсивностей потоков «сырых данных» вводится дополнительные звенья системы постобработки. В них производится фильтрация «интересных» с точки зрения экспериментатора событий на основе системы правил с целью сокращения интенсивности выходного потока данных[7]. Таким образом, исключается избыточная нагрузка на последующие системы обработки. В наиболее общем случае эту систему, примыкающую к постобработчику, можем отнести к классу ГРИД. Как правило, ГРИД системы, подключенные к системам захвата высокоинтенсивных данных, представляют собой сложную многокомпонентную гео-распределенную вычислительную среду [8]. Однако, самыми важными компонентами подобной ГРИД среды являются файловая система и система вычислительных узлов (от англ. Nodes).

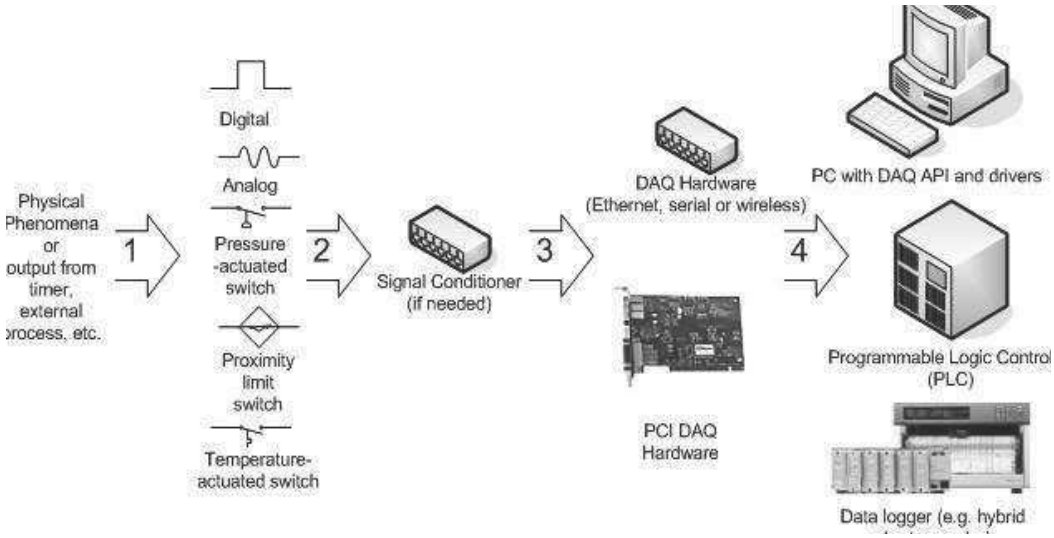


Рисунок 8. Система захвата данных [21]

Две данных подсистемы, управляемые брокером ресурсов представляют собой минимально необходимый базис для организации вычислений. Но этого недостаточно для успешного функционирования среды. Более сложные

среды включают так же т.н. Workflow комплексы, [22] позволяющие не только регистрировать вычислительные задачи, но и управлять планами их выполнения. Необходимо так же обеспечить надлежащий мониторинг вычислительной среды.

Таким образом, мы выяснили, что на сегодняшний день для успешного проведения натурного эксперимента либо вычислительного необходим высокопроизводительный легко масштабируемый программно-аппаратный комплекс способный функционировать в средах с различной степенью гетерогенности и интенсивности потоков данных.

1.4 Обзор современных распределенных файловых систем

Файловая система и в частности – распределенная файловая система является очень важной составляющей любого вычислительного эксперимента [23-25]. Являясь частью операционной системы, она предоставляет необходимый инструментарий для манипулирования устройствами постоянного хранения данных, позволяя абстрагироваться от конкретной природы физических процессов на аппаратном уровне. Современные ядра операционных систем уже снабжены всеми необходимыми модулями для поддержки локальных постоянных хранилищ данных. Клиенты – пользователи ОС могут создавать, удалять, перемещать, читать или записывать файлы. Однако, в отличие от локальных файловых систем, в распределенных - хранилища и клиенты в общем случае распределяются в сети случайно. Фактически получается, что файлы хранятся на разных ресурсах, но отображаются пользователям в виде единой точки доступа. И не смотря на то, что файлы предстают перед пользователем в унифицированном виде, они распределяются между узлами в сети по более сложным алгоритмам. В частности, распределенная файловая система должна быть прозрачной (от англ. Transparency), отказоустойчивой и масштабируемой [23].

- **Прозрачность** предполагает обеспечение надлежащего уровня абстракции для конечного пользователя;
- **Устойчивость** подразумевает способность системы адекватно отрабатывать различные сбои, к примеру, в работе сетевого оборудования;
- **Масштабируемость** подразумевает способность системы оперировать в условиях переменного количества узлов без существенных изменений в качестве обслуживания. Чаще всего количество серверов только возрастает, однако возможна и обратная ситуация. К примеру, для обработки больших данных количество высокопроизводительных машин - серверов может варьироваться от сотен до тысяч единиц.

Исходя из этих свойств, главным преимуществом использования распределенной файловой системы является переносимость и простота интеграции разработанных приложений в контексте ОС. Распределенные файловые системы фактически интегрируются в ОС на уровне API и функционируют без видимых для пользователя отличий от локальных файловых систем. Поэтому, приложение, которое изначально было протестировано на небольшом локальном наборе данных, может быть легко интегрировано в рабочий процесс распределенных вычислений, где оно может оперировать большими наборами. На данный момент, разработано большое количество свободно-распространяемых распределенных файловых систем. Тем не менее, ни одна файловая система (Таб. 2) не является абсолютно универсальной. Большинство распространенных реализаций адаптированы и оптимизированы для вполне определенного класса задач.

Таблица 2 - Сравнительный анализ областей применения систем распределенных файловых систем [23]

Область применения	Файловая система (DFS)
Большие данные	HDFS[26][24] QFS MapR FS
Суперкомпьютинг	Lustre† [27][24] GPFS† OrangeFS BeeGFS Panasas [28]
Многоцелевые	(p)NFS† Ceph† [29][24] Gluster-FS† [30][24] XtreemFS MooseFS iRODS
Специфико-ориентированные	dCache† [31] XRootD† CernVM-FS† [25] EOS†

Так, в своей статье [23] Jacob Blomer замечает, что в физике высоких энергий, к примеру, для хранения файлов могут использоваться сразу несколько различных распределенных файловых систем. Например, XrootD оптимизирован для высокопроизводительного доступа к физическим наборам данных высоких энергий [7-8], [10-12], файловая система Hadoop (HDFS) разработана для инфраструктуры MapReduce файловой системы, а Lustre оптимизирована для высокопроизводительных интенсивно-взаимодействующих приложений на суперкомпьютерах [27]. Вообще говоря, варианты использования различаются как количественном, так и качественном уровнях. Существует несколько публикаций по таксономии, тематическим исследованиям и сопоставлениям производительности распределенных файловых систем [23-25]

1.4.1 Проект GlusterFS

Для проведения исследований в рамках дипломной работы была выбрана файловая система – GlusterFS [30][32]. Она позиционируется разработчиками как высокомасшабируемая децентрализованная файловая система. Важным преимуществом GlusterFS является то, что метаданные не хранятся в едином месте, более того отсутствует сервер метаданных (Таб. 3), который очень часто может стать причиной отказа работы файловой системы. К тому же GlusterFS показал неплохие результаты работы при обслуживании потоков данных различного размера (Таб. 4).

Таблица 3- Сравнительный анализ систем распределенных файловых систем [24]

	HDFS	**iRODS**	**Ceph**	**GlusterFS**	**Lustre**
Architecture	Centralized	Centralized	Distributed	Decentralized	Centralized
Naming	Index	Database	CRUSH	EHA	Index
API	CLI, FUSE, REST, API	CLI, FUSE, API	FUSE, mount, REST	FUSE, mount	FUSE
Fault detection	Fully connect.	P2P	Fully connect.	Detected	Manually
System availability	No failover	No failover	High	High	Failover
Data availability	Replication	Replication	Replication	RAID-like	No
Placement strategy	Auto	Manual	Auto	Manual	No
Replication	Async.	Sync.	Sync.	Sync.	RAID-like
Cache consistency	WORM3, lease	Lock	Lock	No	Lock
Load balancing	Auto	Manual	Manual	Manual	No

[3] WORM principle

GlusterFS разделена на серверную и клиентскую части. На каждом сервере работает служба glusterfsd которая делает доступным для клиентов локальное хранилище в качестве тома. Клиентский процесс glusterfs соединяется с одним или несколькими серверами посредством TCP/IP или InfiniBand и объединяет все доступные серверные тома в один, используя расширяемые трансляторы (функциональные модули системы). Получившийся том монтируется на клиентском хосте при помощи механизма Filesystem in Userspace (FUSE). В своем проекте мы использовали её в контексте TCP/IP стэка для формирования единого параллельного распределенного хранилища данных. [32]

Большая часть функциональности GlusterFS реализована в виде трансляторов (модулей). Использование необходимых трансляторов и их настройка позволяет гибко конфигурировать режим работы системы. Трансляторы реализуют следующую функциональность:

- Синхронная репликация между серверами (нельзя расширить уже существующий том, добавив сервер для репликации);
- Чередование порций данных между серверами (Striping);
- Распределение файлов между серверами;
- Балансировка нагрузки;
- Восстановление после отказа узла (в ручном режиме с помощью опроса файлов (ls -lR или find на смонтированом томе));
- Опережающее чтение (read-ahead) и запаздывающая запись (write-behind) для увеличения быстродействия;
- Дисковые квоты.

Таблица 4 - Сравнительный анализ производительности систем распределенных файловых систем [24], величины – время в секундах

	HDFS		iRODS		Ceph		GlusterFS		MooseFS	
	In	Out	In	Out	In	Out	In	Out	In	Out
1 x 20 GB	407	401	520	500	419	382	341	403	448	385
2 x 20 GB	626	422	1070	468	873	495	426	385	504	478
1000 x 1 MB	72	17	86	23	76	21	59	18	68	4
2 x1000 x 1 MB	96	17	179	20	85	23	86	17	89	4

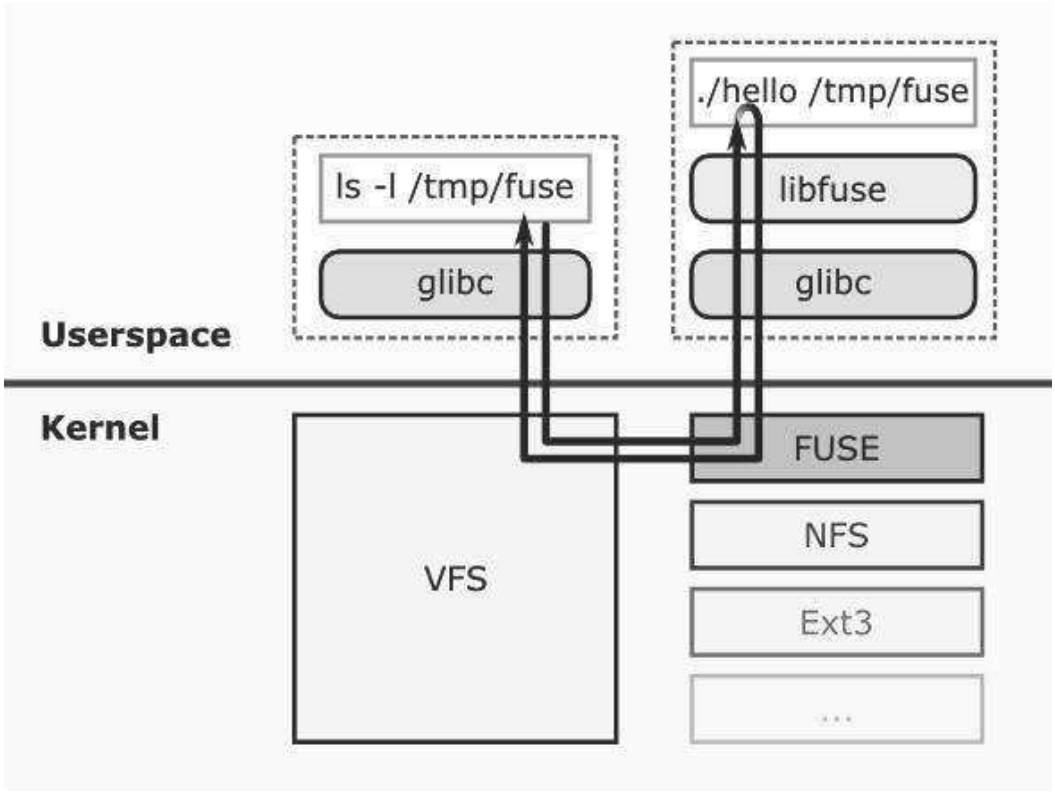


Рисунок 9. Технология FUSE [30]

Важной особенность данной ФС является, то что она основана на базе технологии FUSE и функционирует поверх уже имеющихся ФС (Рис. 9).

1.5 Обзор современных распределенных планировщиков задач

Вторым важным компонентом вычислительной системы является планировщик задач [33]. Ясно, что работа любого эксперимента не мыслима без вычислительных задач и соответствующего им распределения ресурсов. На сегодняшний день, в контексте кластерных архитектур параллельной обработки информации, планировщики заданий являются своеобразными «мета-операционными системами». Планировщики заданий распределяют вычислительные «контейнеры» и контролируют выполнение процессов на этих ресурсах (Рис. 10). В общем смысле термина, планировщик является посредником между приложениями и ресурсами ЦП. К ресурсам обычно относят память и физические устройства. Но центральный процессор (ЦП) можно также считать ресурсом, который планировщик на некоторое время (измеряемое в отрезках) выделяет задаче. Планировщик обеспечивает параллельное выполнение нескольких программ, распределяя ресурсы ЦП между различными задачами различных пользователей.

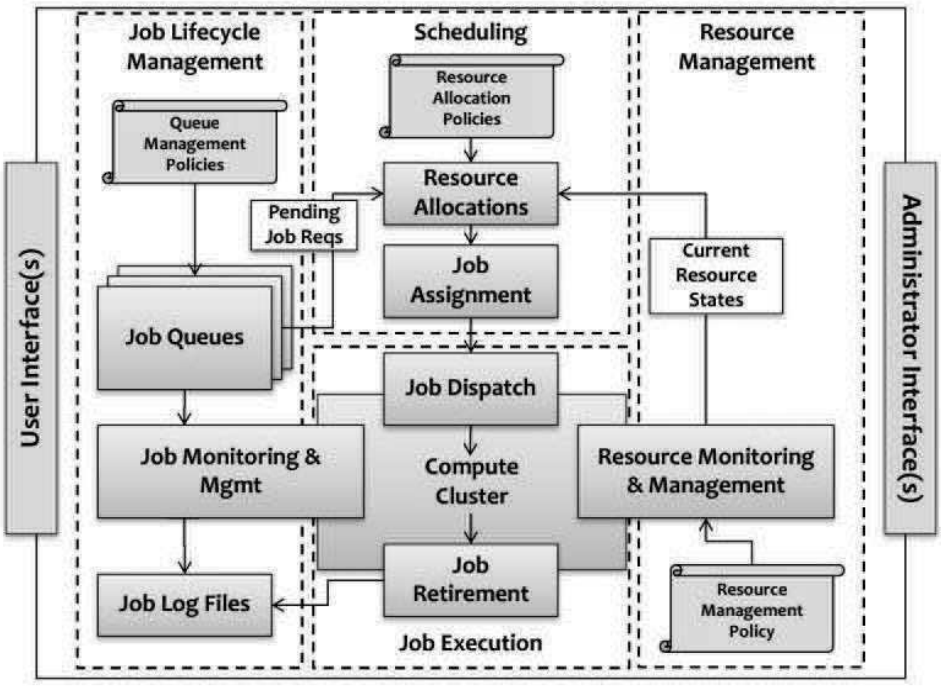


Рисунок 10 Инфраструктура системы распределения задач [33].

Большинство операционных систем, таких как Unix и Windows, предоставляют базовые утилиты для планирования заданий. В частности, с помощью команд(ы) ***at*** и ***batch***, ***cron*** в ОС Linux, можно организовать простую автоматизированную систему распределения задач [34]. Многие программные комплексы, такие как СУБД, также включают в себя соответствующие планировщики.

Важной целью планировщика является эффективное распределение отрезков процессорного времени при условии обеспечения пользователю времени ожидания на приемлемом уровне. Помимо этого, перед планировщиком могут стоять противоречащие друг другу цели, такие, как минимизация времени ожидания при выполнении критически важных задач реального времени и максимальное использование ресурсов ЦП.

Традиционно выделяются две архитектуры, характерные для любой распределенной среды в целом:

- Архитектура Master / Slave(s) - Программное обеспечение Job Scheduler устанавливается на одной машине (Master), тогда как на остальных машинах устанавливается только очень маленький компонент (Клиент), который ожидает команды от Master, выполняет их, а затем возвращает результаты обратно в Master.
- Децентрализованная архитектура - модель, где каждая машина способна участвовать в самоорганизующемся планировании и может перераспределять локально запланированные задания с одного узла на другие взаимодействующие машины. Это позволяет балансировать динамическую рабочую нагрузку, чтобы во-первых максимизировать использование аппаратного ресурса и во-вторых высокую доступность.

В таблице 5 мы приводим краткую сводку актуальных на данный момент систем планирования, выделяем их основные характеристики.

Таблица 5 - Сравнительный анализ систем распределения задач (Batch Schedulers) [35]

	Torque	SLURM	HTCondor	LSF
Developer Support	Maui not supported	Yes	Yes	Yes
Community Support	yes for torque	yes	yes	no
Dcumentation	Good	Good	Good	Good
Licenses/Costs	free	free	free	IBM
Scalability	no	Debated.	yes	6500 nodes
Distribution format	source, rpm	source	source, rpm, tarball	tarball
High Availability	no	head node failover	central manager & job queue failover	head node failover
Stability	low	high	high	high
Installation easiness	yes	without DB yes	yes	yes
Min Num daemons	2 Master / 1 WNs	1 Master / 1 WNs (without DB for fairshares)	4 Master / 2 WNs	±10 master / ± 5 WNs
Queues	yes	no	can use accounting groups to produce similar functionailty	yes
Multicluster	no	yes	yes (flocking, job router)	yes
Requires DB	no	yes for advanced features	no	no
Command line tools	custom commands	custom commands, pbs like wrappers	custom commands	custom commands

1.5.1 Проект SLURM

Для проведения исследований в рамках дипломной работы была выбрана «Простая Linux Утилита для Управления Ресурсами (SLURM)» [33], [36-37] **-** это открытая, надежная и хорошо масштабируемая система управления ресурсами кластера с планировщиком задач, применяемая как для больших, так и для малых Linux-кластеров.

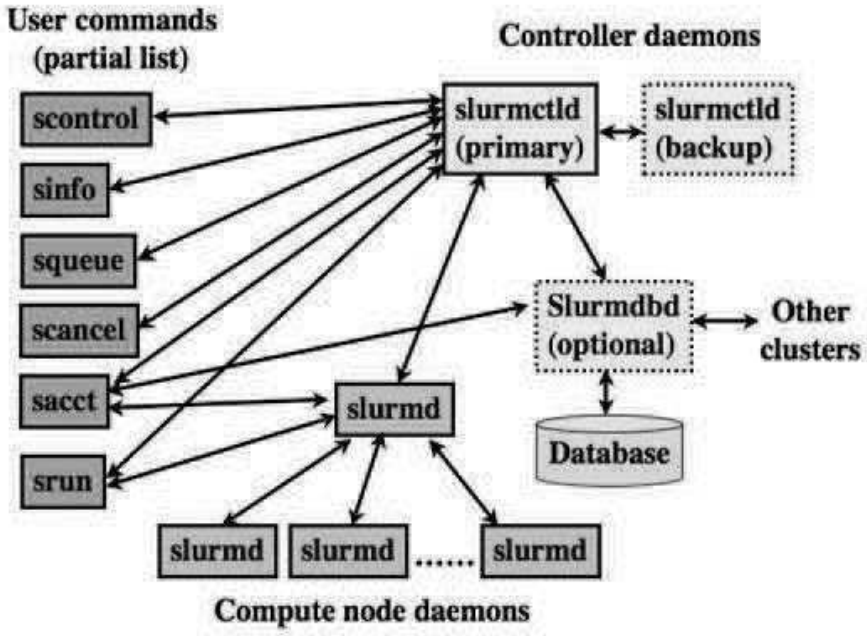


Рис 11. Архитктура SLURM

Как менеджер вычислительных ресурсов кластера, SLURM выполняет три ключевых функции.

- Во-первых, она определяет выделенный и/или совместный доступ пользователей локальным ресурсам (вычислительным узлам) на некоторое время для выполнения ими вычислительных задач.

- Во-вторых, он обеспечивает функционирование структуры запуска, выполнения и мониторинга задач (обычно это параллельные задачи) на выделенных узлах.

- Наконец, распределяет ресурсы, управляя очередью ожидающих запуска задач.

SLURM состоит (Рис. 11) из сервиса slurmd, запускающегося на каждом вычислительном узле, и центрального сервиса slurmctld, запускающегося на управляющем узле (опционально - с резервной копией управляющего узла).

Сервисы slurmd образуют отказоустойчивую иерархическую структуру. Пользовательские команды включают: sacct, salloc, sattach, sbatch, sbcast, scancel, scontrol, sinfo, smap, squeue, srun, strigger и sview. Все эти команды могут быть запущены как с управляющего сервера, так и с узлов кластера.

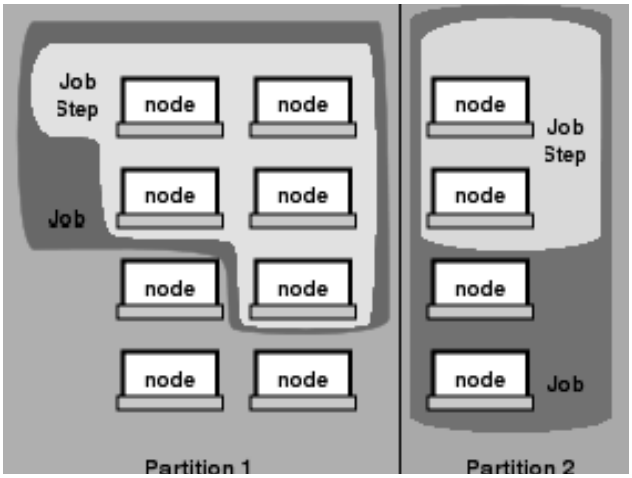


Рисунок 12. Связи между всеми узлами сети в рамках планировщика SLURM

Объекты, управляемые сервисами SLURM, это узлы - вычислительный ресурс SLURM-а, разделы, которые объединяют узлы в логические (возможно, частично перекрывающихся) множества. Задачи подмножества ресурсов, выделенных пользователю на указанное количество времени, и шаги задачи, которые являются множествами (возможно параллельных) подзадач в рамках задачи (Рис. 12).

Разделы могут рассматриваться как очереди задач, каждая из которых имеет комплекс ограничений, как например ограничение задачи по размеру, по времени выполнения, уровню доступа пользователей и т. д. Задачи в очереди упорядочены по приориету и им выделяются ресурсы в соответствующем разделе.

Как только для задачи выделено множество узлов, пользователь может запускать параллельную работу в виде шагов задачи в любой конфигурации, в пределах выделенных узлов.

Важным преимуществом является то, что обработка данных с помощью SLURM не ограничивается какой-то определенной парадигмой, к примеру, MapReduce, предлагаемой платформой Hadoop по умолчанию. При работе с данным программным обеспечением для планирования задач в многозадачной распределенной среде отсутствуют крупные интеграционные проблемы.

1.6 Добровольные вычисления. Информация о системе BOINC

В первоначальной конфигурации системы, разработанной в рамках выпускной квалификационной работы бакалавра использовалась система BOINC (Berkeley Open Infrastructure for Network Computing) [38-39]— свободная (лицензия LGPL) программная платформа, разработанная университетом Беркли, для организации распределённых вычислений, фактически реализующая модель волонтерского ГРИДа. Распределённые вычисления в рамках волонтерской модели позволяют получить огромную вычислительную мощность, соединяя в сеть множество отдельных компьютеров, на которых и проводятся нужные вычисления. Вычислительная мощность у популярных BOINC проектов, таких как, SETI@home, сравнима с производительностью суперкомпьютеров, конечно же, с некоторой оговоркой - только в рамках определенного класса задач. Это позволяет научным (в подавляющем большинстве случаев) и коммерческим проектам проводить ресурсоёмкие расчёты за короткие сроки без использования дорогостоящего оборудования.

Так же CERN, в лице различных отделов, по мимо использования специализированного ПО для WLCG, реализует ряд волонтерских проектов под общим названием LHC@home. Среди них ATLAS@home, Beauty, CMS@home, SixTrack и Test4Theory (Рис 13). Подробная информация обо всех проектах программы LHC@home может быть найдена на официальном сайте [40].

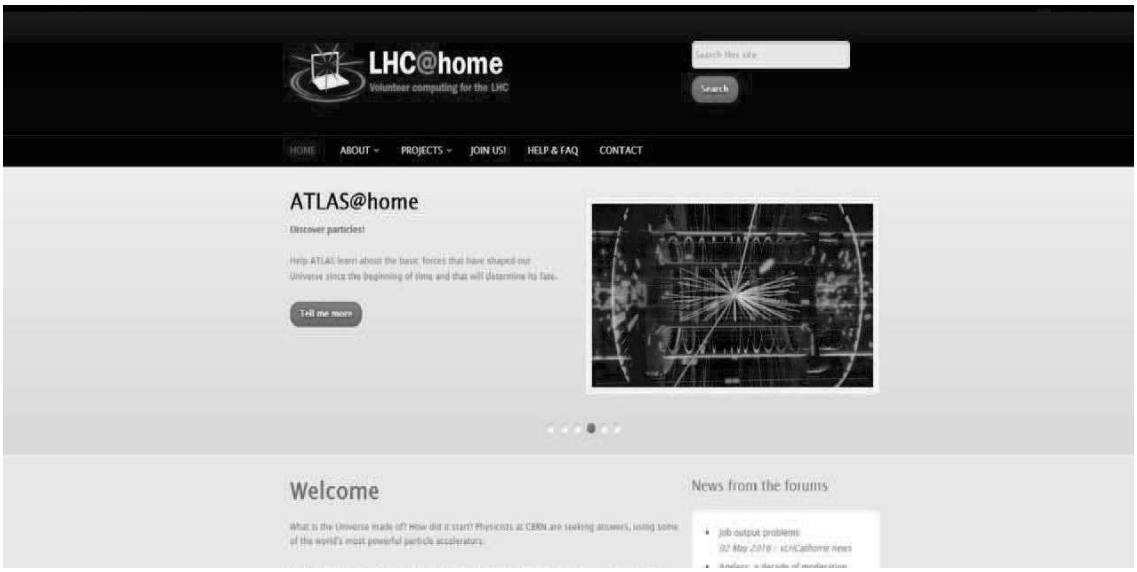


Рисунок 13. Официальный ресурс LHC@home

Любой владелец физического или виртуализированного «компьютера», либо в особом случае, вычислительной фермы, может присоединиться к проекту с помощью программы-клиента BOINC и помочь ему, жертвуя часть ресурсов (HDD, RAM, CPU, GPU) своего компьютера/фермы для вычислений в рамках определенного проекта. BOINC состоит из серверной части (для организаторов проекта) и клиентской части (программа, с которой работает участник проекта). Клиентская часть состоит из так называемого BOINC клиента и графической программы для мониторинга работы BOINC клиента, которая называется BOINC менеджер (BOINC Manager).

После сборки из исходного кода/загрузки через пакетный менеджер BOINC клиента пользователь может выбрать, в каких проектах ему принять участие и присоединиться к выбранным (Рис. 14), копируя URL адрес в меню **BOINC Manager**. К тому же, те же самые действия и другие операции над клиентом можно производить с помощью командой строки, благодаря клиенту **boinccmd**, реализующему RPC управление через Unix Socket, Domain Socket. Особо опытные Linux пользователи администрируют собственные вычислительные фермы как раз с помощью данной утилиты. После подготовки клиента и загрузки заданий (одного или нескольких

исполняемых модулей), запустятся вычисления в рамках поступивших входных данных, а после окончания вычислений, результаты обработки входных данных исполняемым модулем будут отправлены обратно на сервера проектов. Вся работа BOINC клиента будет заключаться в этом цикле.

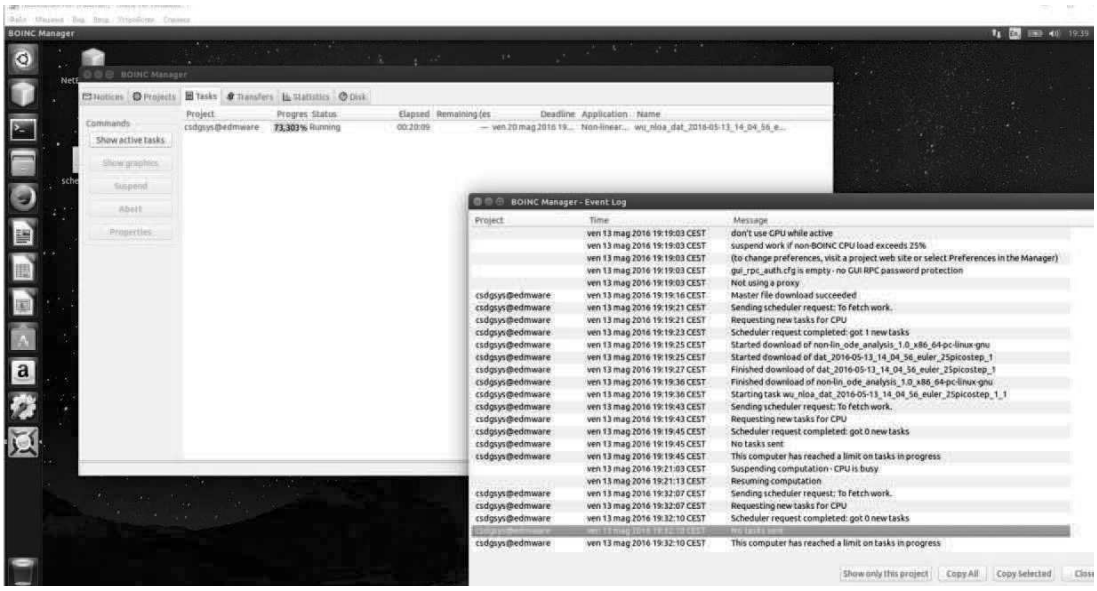


Рисунок 14. Инициализация окружения в рабочем цикле (Linux GUI)

Из репозитория (Download) проекта загружается задание, затем выполняется его обработка (математические расчёты, файловый поиск), и результат отправляется обратно на сервер проекта, в репозиторий (Upload). Важно заметить, что **BOINC использует только свободные ресурсы вычислительной мощности компьютера**, поэтому можно спокойно продолжать пользоваться компьютером в обычном режиме, даже при включённом BOINC клиенте. Обычно большую часть времени на персональных компьютерах ресурсы не используются полностью (большую часть времени процессор не загружен на 100%). Именно эти свободные промежутки будет использовать BOINC для обработки заданий. Также пользователи BOINC часто не выключают компьютеры на ночь, предоставляя это время для работы BOINC. Распространена так же практика организации отдельных виртуализированных (облачных), либо реальных

вычислительных ферм. Стоит обратить внимание на то, что активное использование BOINC может увеличить потребление электроэнергии и привести к перегреву центрального процессора. Поэтому помимо разовой регистрации и предоставления собственных ресурсов проектам необходим постоянный контроль над машинами и сбор соответствующей статистики.

ГЛАВА 2. РАЗРАБОТКА

2.1 Техническое задание

Ниже представлен документ на проектирование технического объекта . Он устанавливает основное назначение разрабатываемого объекта, его технические характеристики, показатели качества и технико-экономические требования, предписание по выполнению необходимых стадий создания документации (конструкторской, технологической, программной и т. д.) и её состав, а также специальные требования.

2.1.1 Термины и сокращения

Сеть – система связи компьютеров или вычислительного оборудования (серверы, маршрутизаторы и другое оборудование).

Распределенные вычисления – Распределённые вычисления — способ решения трудоёмких вычислительных задач с использованием нескольких компьютеров.

GRID, Грид – это набор сервисов и вычислительных узлов, объединенных между собой для решения единой вычислительно-емкой задачи.

Сервис – это компьютерная программа, работающая в фоновом режиме без интерактивного взаимодействия с пользователем.

СУБД –совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Сервер –специализированный компьютер и/или специализированное оборудование для выполнения на нём сервисного программного обеспечения.

Файловая система – совокупность сервисов, предоставляющая надлежащий уровень абстракции в контексте ОС для доступа к физическим устройствам хранения данных.

Распределенная файловая система - файловая система распределенная в сети на произвольном количестве вычислительных устройств с различной степенью связности и гетерогенности.

Вычислительный контейнер – совокупность алгоритмов, приведенных к стандарту исполняемого типа в контексте данной вычислительной среды;

Планировщик задач - совокупность сервисов, предоставляющая надлежащий уровень абстракции в контексте ОС для доступа манипулирования вычислительными контейнерами.

2.1.2 Общие сведения

Полное наименование системы:

CSDG@home - сеть типа GRID для организации высокопроизводительных масштабируемых научных вычислений в гетерогенных средах.

Заказчик:

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «ЮУрГУ», Высшая Школа

Электроники и Компьютерных Наук, Кафедра «Электронные вычислительные машины» - г. Челябинск, пр-к Ленина, 76

Исполнитель работ и услуг:

Кашанский В.В., студент магистратуры ВШЭКН ЮУРГУ

Плановые строки начала и окончания работы:

Дата начала работ: 25.01.2018 г.

Дата окончания работ: 31.05.2018 г.

Нормативные документы:

ГОСТ 34.602-89 Техническое задание на создание автоматизированной системы

2.1.3 Назначение и цели создания системы

Назначение системы:

Сеть типа GRID для организации распределённых вычислений различной производительности предназначена за счет использования выделенных и незадействованных вычислительных мощностей кафедры ,а так же ресурсов заинтересованных в результате лиц.

Цели создания системы:

Целью создания данной системы является решение задач численного моделирования систем уравнений и анализа данных, а также использование в курсах обучения распределенным вычислениям.

Классы решаемых системой задач:

1. Задача параллельного интегрирования (с точностью до 10^{-6});
2. Имитационное моделирование, численное решение ДУЧП (расчет полей при заданных параметрах распределения) (с точностью до 10^{-6});

3. Анализ экспериментальных данных, файловый поиск, обработку сигналов (FFT), изображений (объем исходных данных хранящихся на сервере до 10 Гб);

2.1.4 Требования к системе

Требования к структуре и функционированию системы:

Для реализации данной системы необходим следующий набор подсистем:

1. **HTTP-сервер Apache** с веб-интерфейсом в виде набора **PHP-скриптов**. С помощью веб-сайта обеспечивается общее управление системой, создание, конфигурация, управление и получение результатов заданий и регистрация новых пользователей;
2. **MySQL** база данных, обеспечивающая хранение данных о пользователях, заданиях;
3. **Набор сервисов** (генератор заданий, файловая система, планировщик, валидатор, ассимилятор результатов), обеспечивающих организацию взаимодействия с пользователем и данными, выдача, получение ассимиляция результатов;
4. **Клиентское приложение под различные платформы** – организация выполнения вычислений за счет ресурсов пользователя, получение задания от сервера и отправка результатов на сервер;
5. **Графический интерфейс клиентского приложения** – настройка и контроль работы клиентского приложения, визуальное отображение регистрации в системе, получения задания и прогресса их выполнения;
6. **Графический интерфейс администратора -** настройка и контроль работы вычислительной системы.

Требования к функциям, выполняемым системой:

Система должна иметь следующий функционал:

1. **Регистрация и контроль узлаов сети.** Пользователи должны иметь возможность зарегистрироваться в системе BOINC, а так же зарегистрировать список всех доступных машин для своей учетной записи. Так же должна быть возможность присоединения узлов к локальной вычислительной системе.
2. **Распределение задачи между узлами сети.** Администратор проекта должен иметь возможность гибкого распределения задач в имеющемся вычислительном пуле.
3. **Получение результата вычислений узлами сети.** Система должна корректно отрабатывать как на входном, так и на выходном массивах данных. Целостность информационного потока должна сохранятся.
4. **Агрегирование результатов работы сети.** После обработки выходного массива данных, полученного от различных вычислительных заданий, система обязана сформировать файл по определенному правилу, задаваемому администратором проекта.
5. **Формирование отчета по результатам вычислений**. Файл-отчет сформированный в удобном для человека формате должен быть обязательно передан определенному лицу либо списку лиц, указанных администратором.

Требования к информационной безопасности:

В системе должна быть предусмотрена защита от следующих типов угроз безопасности:

1. Подделка результатов вычислений;
2. Подделка времени выполнения;
3. Заражение вирусом результата вычислений;
4. Заражение вирусом распространяемого задания;
5. Умышленное переполнение базы данных сервера;
6. Кража данных учетной записи с сервера;
7. Перехвата передаваемых данных учетной записи.

8. Кража файлов проекта;
9. Кража файлов участников сети;
10. Злоупотребление ресурсами участников сети.

Проблемы в пунктах 3,4 решаются путем генерирования цифровых подписей для исполняемых модулей.

Проблемы в пунктах 1,2,5,6,8,9,10 решаются с помощью анализа журналов системы контроля версий, поиска уязвимостей и установки stable релизов ПО и ядра ОС.

Проблемы в пункте 7 решаются путем шифрования SSL информационного потока (RSA, DSA, AES, DES).

2.1.5 Требования к аппаратной части системы

Опираясь на примерный масштаб заданий описанных в техническом задании для узлов файловой системы GlusterFS был выбран сервер-экземпляр со следующими характеристиками:

1) IntelCore 2 DuoProcessorE4300 1.80 GHz;
2) SDRAM DDR21 Гб;
3) HDD 75 Гб;
4) Канал между сервером и интернетом не менее 100 Мбит\с.

Для системы BOINC был выбран сервер со следующими характеристиками:

5) IntelCore 2 DuoProcessorE4300 1.80 GHz;
6) SDRAM DDR21 Гб;
7) HDD 30 Гб;
8) Канал между сервером и интернетом не менее 100 Мбит\с.

Для узлов SLURM был выбран сервер-экземпляр со следующими характеристиками:

1) Intel Core I7-4770 3.40 GHZ
2) SDRAM DDR3 4 Гб;
3) HDD 30 Гб;
4) Канал между сервером и интернетом не менее 100 Мбит\с.

Исходя из возможной вычислительной сложности потенциальных задач и приемлемой вычислительной мощности сети, клиентские машины (участники сети) должны удовлетворять следующим минимальным требованиям:

1) Одноядерный ЦП архитектуры x86/x64 с частотой 2 ГГц или выше\с большим количеством ядер;
2) ОЗУ 512 Мб или больше;
3) 1 Гб или больше дискового пространства;
4) Канал между сервером и интернетом не менее 10 Мбит\с.

2.1.6 Требования к программному обеспечению системы

Основываясь на информации представленной в предыдущих пунктах составим требования к ПО, которое должно быть установлено на сервере BOINC:

1) Операционная система Linux Centos 6 с ядром версии 2.6.32 или выше и поддержкой EPOLL
2) Сервер BOINC;
3) СУБД MySQL;
4) Web-сервер Apache;
5) Интерпретаторы и стандартные библиотеки языков python и php.

На клиентские машины должно быть установлено следующее ПО:

1) Операционная система Linux с ядром версии 2.6.32, либо WindowsXP или выше, либо другая ОС из списка поддерживаемых конкретным приложением в рамках проекта;
2) Программный пакет (клиентское приложение + графический интерфейс) BOINC 7.x или выше.

Основываясь на информации представленной в предыдущих пунктах составим требования к ПО, которое должно быть установлено на сервере GusterFS и SLURM:

1) Операционная система Linux Centos 7 с ядром версии 3.10 или выше
2) Сервер SLURM (slurmctl, slurmd)
3) Сервер и клиент GlusterFS(glusterfs, glusterfsd)

2.1.7 Технико-экономические показатели

Принято решение, что в качестве сервера BOINC будет выступать арендованная в ДЦ виртуальная машина с идентичными характеристиками, выделенным IPv4 адресом, высокопропускным каналом связи. Это позволит привлекать не только локальные, но и глобальные ресурсы, сделав систему открытой.

Цена в месяц: 378 р/мес за 1 сервер. На данный момент в обеспечении требуемой инфраструктуры участвуют 3 сервера. Один обеспечивает функционирование системы BOINC и ряда служебных сайтов. Другой - почтовый. Итого 1134 р/мес за сервера.

Так же для проекта приобретено доменное имя, обеспечивающее независимость уровней OSI 7 и OSI 3. Оно позволяет изолировать физический уровень от логического, предоставляя удобное для запоминания имя. Стоимость доменного имени 920 р/год. ПО является свободным и не требует покупки.

Остальные ресурсы для файлового хранилищаGlusterFS и вычислительных узлов SLURM предоставлены СКЦ ЮУрГУ.

2.1.8 Состав и содержание работ по созданию системы

1) Ознакомление с требованиями к аппаратной и программной части;
2) Установка ОС на сервера;
3) Установка необходимого ПО на сервера;
4) Конфигурирование вспомогательного ПО;
5) Проектирование баз данных;
6) Сборка и конфигурация серверов BOINC, GlusterFS, SLURM;
7) Запуск серверов и соответствующих служб;
8) Тестирование;
9) Приемочные испытания;
10) Запуск в эксплуатацию.

2.1.9 Порядок контроля и приемки системы

К началу приемки системы должны быть предоставлены:

- программа приемки системы;
- инструкция по эксплуатации.

Приемка будет осуществляться комиссией, состоящей из компетентных в данной области специалистов, назначенных Заказчиком и ответственных за обслуживание программно-аппаратного комплекса. Тестирование программной платформы будет осуществляется во время приемки, на основе тестовых заданий, предварительно подготовленных Исполнителем, для каждого описанного класса задач. По результатам приемки составляется акт приемки-сдачи системы в эксплуатацию.

2.1.10 Требования к документированию и гарантийное сопровождение

Система должна поставляться со следующей документацией:

1. Общее описание информационной системы;
2. Руководство пользователя;
3. Руководство администратора системы.

В случае выявления недостатков в программах и разработках, составляющих результаты работ, исполнитель должен по требованию Заказчика незамедлительно (в срок не более 5 календарных дней) устранить такие недостатки. Гарантийный срок продлевается на период устранения недостатков.

ГЛАВА 3 РЕАЛИЗАЦИЯ СИСТЕМЫ

3.1Структура системы

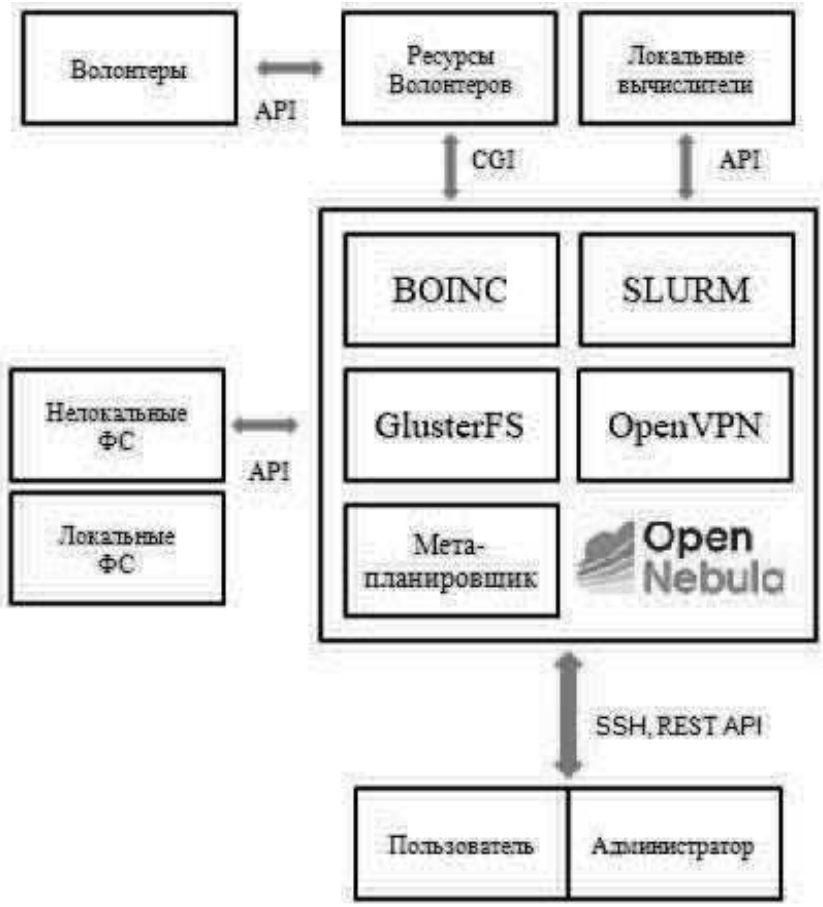


Рисунок 15. Диаграмма конечной ГРИД системы

ГРИД система состоит из набора отдельных распределенных подсистем (Рис. 15), каждая из которых отвечает за свою вполне определенную задачу, например, выполнение вычислений, хранение файлов и т.д. Каждая из подсистем проверяет состояние подзадачи, производит какие-то действия и изменяет состояние подзадачи – так они работают в бесконечном цикле. В целом, система состоит из:

- **сервера BOINC**, при необходимости распределенного на несколько физических серверов в целях повышения производительности, отказоустойчивости и безопасности;
- **сервера GlusterFS** распределенного на множестве узлов;
- **сервера SLURM** с централизованным управлением;
- **множества клиентов BOINC и SLURM**, выполняющих задания сервера;
- дополнительных компонент в виде присоединенных GRID-сетей.

Необходимые локальные ресурсы обеспечиваются системой **OpenNebula**.

3.1.1 Служба обработки состояния подзадач (Transitioner)

Данная служба BOINC предназначена для обработки состояния вычислительных подзадач и результатов их решения. Она не зависит от приложения и является одинаковой для всех проектов. Служба обработки проверяет текущее состояние подзадачи в базе данных и обновляет соответствующие поля, когда подзадача готова перейти в новое состояние.

Служба обработки является одной из наиболее ресурсоемких с точки зрения нагрузки на процессор, поэтому она может быть разделена на несколько процессов, каждый из которых отвечает за определенный набор подзадач. Соответственно, эти процессы могут работать на различных физических серверах.

3.1.2 Служба проверки результатов (Validator)

Назначение данной службы BOINC – организация проверки полученных результатов. В целях обеспечения достоверности каждая из подзадач рассчитывается на нескольких различных клиентах. Поэтому полученные в итоге результаты необходимо проверить, сверив между собой и определив «каноническое» решение. Алгоритм проверки результатов целиком зависит

от решаемой задачи. Кроме того, программа проверки следит за правдоподобностью результатов.

При запуске служба обращается к базе данных для получения информации о новых результатах, требующих проверки. Если таковые есть, то служба проверки запускает соответствующую функцию сравнения результатов. Для каждой глобальной задачи, решаемой с помощью распределенных вычислений на базе BOINC, необходимо разработать две функции в рамках службы проверки: одна функция сравнивает два результата, а другая – наборы результатов. Первая используется для принятия решения о предоставлении очков, когда клиентским приложением передан новый результат и найдено каноническое решение. Вторая функция используется для определения самого канонического результата из набора результатов, переданных несколькими клиентами. Количество результатов, необходимых для выработки канонического решения, определяется при создании подзадачи. Это значение может быть установлено для всего приложения в целом, но также возможно указать различные значения для различных клиентов.

3.1.3 Служба освоения (Assimilator)

Программа освоения периодически проверяет наличие завершенных подзаданий. Администратор проекта должен создать функцию, которая определяет, что необходимо сделать с каноническими результатами. Например, их можно архивировать и пересылать по электронной почте определенному человеку или автоматически запускать последующую обработку данных, выделяя интересующие фрагменты и записывая их на устройство хранения. Подзадача помечается как завершенная только после прохождения через службу освоения.

3.1.4 Служба удаления файлов (Filedeleter)

Служба удаления файлов – это «сборщик мусора» проекта BOINC, она просто проверяет наличие завершенных и освоенных подзадач, а при нахождении таковых очищает входные и выходные файлы сервера, связанные с этими подзадачами. Необходимо, чтобы выходные файлы, содержащие канонический результат, где-то хранились на фазе освоения. При необходимости удалять можно только файлы, оставляя записи в базе данных, тогда всегда можно будет пройти по базе данных и найти информацию о подзадачах, результатах и т.д. даже после завершения подзадач (и удаления файлов).

3.1.5 Служба подачи (Feeder)

Служба подачи является вспомогательной – она загружает необработанные подзадачи из базы данных в сегмент разделяемой памяти. Эта предварительная работа выполняется для повышения производительности системы BOINC в целом с помощью ограничения числа запросов к базе данных.

3.1.6 Файловая система (GlusterFS)

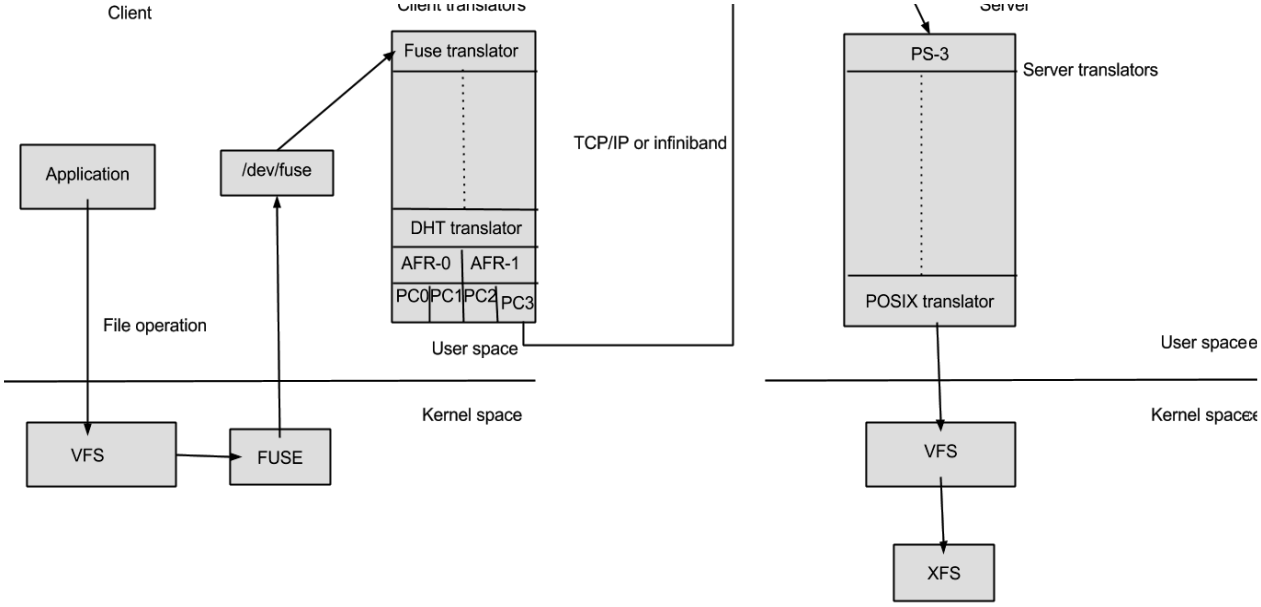


Рисунок 16. Взаимодействие Клиент-Сервер (glusterfs-glusterfsd)

На каждом сервере работает служба glusterfsd которая делает доступным для клиентов локальное хранилище в качестве тома. Клиентский процесс glusterfs (Рис. 16) соединяется с одним или несколькими серверами посредством TCP/IP или InfiniBand и объединяет все доступные серверные тома в один, используя расширяемые трансляторы (функциональные модули системы). Получившийся том монтируется на клиентском хосте при помощи механизма Filesystem in Userspace (FUSE). В своем проекте мы использовали её в контексте TCP/IP стэка для формирования единого параллельного распределенного хранилища данных. Данные из смонтированного тома могут быть в дальнейшем использованы волонтерскими, либо локальными вычислителями.

3.1.7 Планировщик (Scheduler)

Планировщик – это программа, которая запускается, когда к серверу проекта подсоединяется клиент и запрашивает порцию заданий, либо когда пользователь желает добавить вычислительную задачу. В контексте нашей системы следует различать три планировщика BOINC, SLURM и мета-планировщик. Планировщик BOINC получает задания (Рис. 17) из сегмента разделяемой памяти, в который задания загружает служба подачи. Мета-планировщик используется для распределения заданий между системой BOINC и SLURM. Мета-планировщик имеет возможности по самостоятельному назначению подзадач системам BOINC и SLURM, так как не все задачи имеют одинаковые параметры, удовлетворяющие той или иной модели вычислений.

Во время сессии работы с планировщиком клиент также отчитывается о завершенных работах, которые были уже загружены на сервер со времени последней сессии планирования. В итоге клиент остается со списком заданий на обработку и списком адресов, с которых можно получить необходимые файлы, т.е. входящие файлы и файлы приложения, если их нет на клиентском узле.

3.1.8 Мост (Bridge)

Данная служба, обеспечивает связь и совместную работу над проектом инфраструктуры BOINC и стандартной GRID. Приложения BOINC специально разрабатываются под архитектуру BOINC. Они вызывают функции BOINC через интерфейсы, реализованные в клиенте и выполняющие такую специфическую работу как, например, разрешение имен файлов. Как следствие, подзадачи проекта BOINC не могут быть напрямую (без дополнительных модификаций) запущены для расчета на инфраструктуре Grid. С другой стороны, Grid не может, подобно клиенту BOINC, без посредника соединиться с планировщиком проекта и запросить подзадачи для расчета. Такие проблемы взаимодействия различных

архитектур распределенных вычислений требуют реализации дополнительных механизмов, делающих возможной связку BOINC-Grid. Эти задачи и решает программный мост, при этом фактическая реализация моста может зависеть от особенностей подключаемой Grid и проекта, в рамках которого проводятся вычисления.

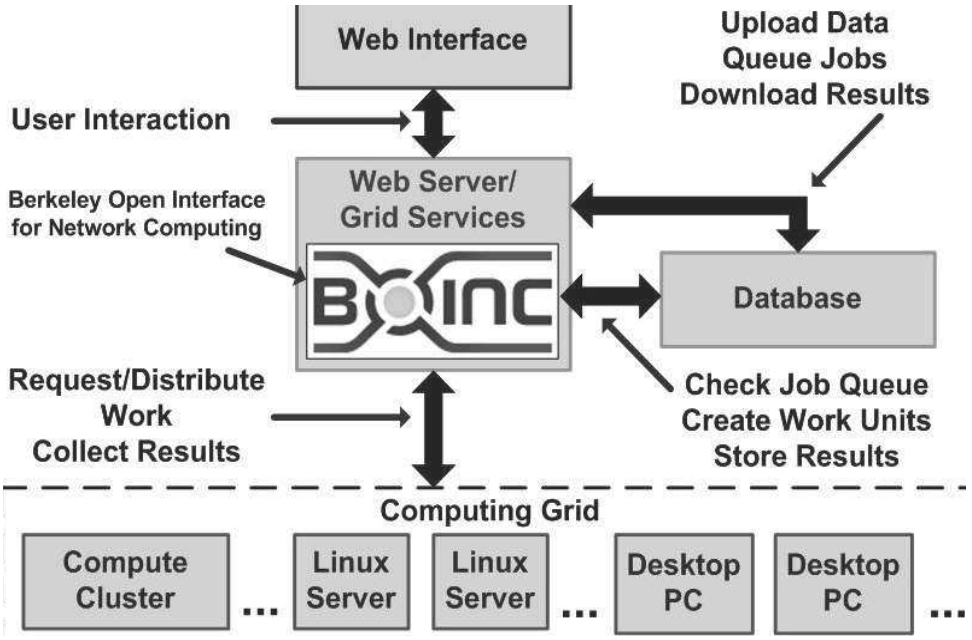


Рисунок 17. Процесс взаимодействия отдельных компонент BOINC

ГЛАВА 4. ВНЕДРЕНИЕ И ЭКСПЛУАТАЦИЯ СИСТЕМЫ

4.1 Развертывание системы

4.1.1 BOINC

BOINC клиент и менеджер доступны для GNU/Linux, Mac OS X, Windows, Android систем, распространяются под свободной лицензией LGPL. Для пользователей KDE доступен альтернативный менеджер на Qt — KBoincSpy. Более подробную информацию о получении, установке и использовании этих программ можно найти на официальном сайте BOINC. Пользователи Windows могут легко установить клиент, скачав лишь исполняемый файл установщика. Пользователи Linux могут так же воспользоваться менеджерами пакетов для установки ПО.

К примеру, в Ubuntu выполнить следующую команду:

```
sudo aptitude install boinc-client boinc-manager
```

Либо загрузить исходный код из репозитория GIT (Github) - https://github.com/BOINC/boinc и скомпилировать вручную. Данный вариант является наиболее дистрибутиво-независимым, требуя только минимально необходимого окружения для сборки проекта. Сервер собирается отдельно из исходных кодов путем указания соответствующих параметров для утилиты make. Так, к примеру, в системе CentOS 7.0 (Minimal) для компиляции клиента (boinc, boinccmd) и сервера потребуется следующий стек ПО:

- gcc-c++
- autoconf
- openssl-devel
- automake
- libtool
- libcurl-devel

Все ПО по умолчанию доступно в репозитории CentOS (base) и его можно получить выполнив следующую команду:

```
yum -y install gcc-c++ autoconf openssl-devel automake libtool libcurl-devel
```

Порт 80 требуется открыть на узле сервера.

4.1.2 GlusterFS

GlusterFS клиент и сервер доступны для GNU/Linux, распространяются под свободной лицензией . Загрузить данное в системе Centos 7 ПО можно из репозитория The CentOS Storage Special Interest Group - centos-gluster312. Либо загрузить исходный код из репозитория GIT (Github) - https://github.com/gluster/glusterfs и скомпилировать вручную. Данный вариант является наиболее дистрибутиво-независимым, требуя только минимально необходимого окружения для сборки проекта

Далее, для работы glusterfs необходимо открыть следующие порты на всех узлах:

Основные:

- 24007 TCP;
- 24008 TCP;

portmapper:

- 111 TCP,UDP

Один порт на каждый brick:

- 49152 TCP;
- 49153 TCP;

```
yum install glusterfs-server
chkconfig glusterd on && service glusterd start
gluster peer probe gluster02.local
gluster peer status

Number of Peers: 1
```

```
Uuid: 114bab47-15d9-446a-bf12-123qea0cd3fa
State: Peer in Cluster (Connected)
Hostname: gluster02.local
[gluster02.local]# gluster peer status

Number of Peers: 1

Hostname: gluster01.local
Port: 24007
Uuid: 73aa5a33-a0a0-4ffd-9qac-5456e3qc16b7
State: Peer in Cluster (Connected)
Gluster volume create vlm0 replica 2 gluster01.local:/data/brick1/vlm0
gluster02.local:/data/brick1/vlm0
gluster volume start vlm0
```

4.1.3 SLURM

SLURM клиент и сервер доступны для GNU/Linux, распространяются под свободной лицензией. Загрузить исходные коды можно загрузить с официального сайта, либо из репозитория GIT (Github) - https://github.com/SchedMD/slurm и скомпилировать вручную. К примеру, в системе Ubuntu установка выполняется путем выполнения запроса к менеджеру пакетов.

```
sudo apt-get install slurm-llnl
```

Для системы CentOS необходимо загружать исходные файлы вручную либо из репозитория, либо из официального сайта.

Для работы необходимо открыть следующие порты на всех узлах:

- SlurmctldPort=6817
- SlurmdPort=6818

Далее, на сайте https://slurm.schedmd.com/configurator.html необходимо собрать конфигурацию конечной системы.

```
# slurm.conf file generated by configurator.html.
```

```
# Put this file on all nodes of your cluster.
# See the slurm.conf man page for more information.
#
ControlMachine=linux-cl
#
AuthType=auth/none
CacheGroups=0
CryptoType=crypto/openssl
MpiDefault=none
ProctrackType=proctrack/pgid
ReturnToService=1
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmctldPort=6817
SlurmdPidFile=/var/run/slurmd.pid
SlurmdPort=6818
SlurmdSpoolDir=/tmp/slurmd
SlurmUser=slurm
StateSaveLocation=/tmp
SwitchType=switch/none
TaskPlugin=task/none
#
# TIMERS
InactiveLimit=0
KillWait=30
MinJobAge=300
SlurmctldTimeout=120
SlurmdTimeout=300
Waittime=0
#
# SCHEDULING
FastSchedule=1
SchedulerType=sched/backfill
SelectType=select/linear
#
# LOGGING AND ACCOUNTING
AccountingStorageType=accounting_storage/none
ClusterName=cluster
JobCompType=jobcomp/none
JobCredentialPrivateKey = /usr/local/etc/slurm.key
JobCredentialPublicCertificate = /usr/local/etc/slurm.cert
JobAcctGatherFrequency=30
JobAcctGatherType=jobacct_gather/none
SlurmctldDebug=3
SlurmdDebug=3
#
# COMPUTE NODES
NodeName=linux[1-32] CPUs=1 State=UNKNOWN
PartitionName=debug Nodes=linux[1-32] Default=YES MaxTime=INFINITE State=UP
```

Результат выполнения команды sinfo:

```
$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug*       up   infinite      1   idle linux-cl
$
```

4.2 Эксплуатация системы. Жизненный цикл задания.

Жизненный цикл задания и связанных с ним подзаданий выглядит следующим образом (Рис. 18). Программа генерации заданий (*мета-планировщик*) создает подзадачи и необходимые для их расчета входные файлы. Далее, мета-планировщик создает для каждого из подзаданий один или несколько одинаковых экземпляров (в зависимости от настроек проекта), затем распределяет подзадания различным клиентским программам (BOINC, SLURM).

Распределение происходит между узлами со схожим с установленным в задании значением требуемой вычислительной мощности (в FLOPS\MIPS), а так же сетевой связности. Далее каждая клиентская программа загружает с сервера входные файлы. После расчета подзадания клиентская программа загружает выходные файлы на сервер; клиентская программа отчитывается о выполнении подзадания (возможно, после небольшой задержки, необходимой для снижения нагрузки на программу-планировщик сервера). Служба проверки результатов (*разработанная специально для проекта*) проверяет выходные файлы, возможно, сравнивая результаты различных клиентских программ.

Когда найдено каноническое решение, служба освоения (*разработанная специально для проекта*) обрабатывает результаты, например, помещая их в отдельную базу данных или отсылая на электронную почту. Когда все экземпляры подзадания завершены, служба удаления файлов удаляет ненужные больше входные и выходные файлы, а также очищает базу данных от информации о подзадании и его экземплярах.

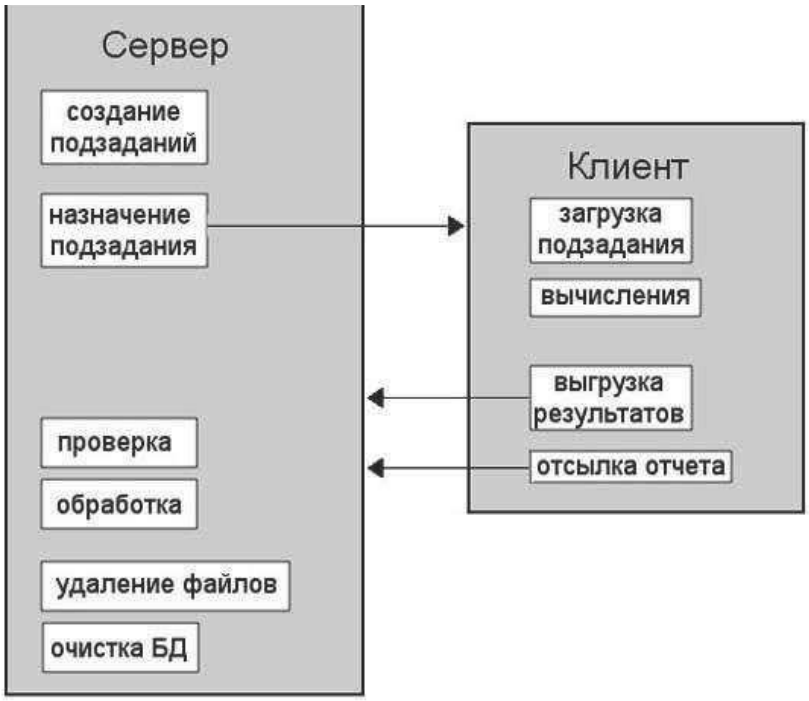


Рисунок 18. Жизненный цикл заданий в системе

Обратим внимание на то, что некоторые службы являются стандартными и не зависят от конкретного проекта и его реализации. Однако другие службы необходимо разрабатывать отдельно для каждого проекта – в этом и состоит дополнительная сложность, которой приходится расплачиваться за возможность проведения распределенных вычислений.

4.3 IDEF диаграммы взаимодействия с системой

После внедрения системы путем незначительного усложнения этапов подготовки задания удалось автоматизировать функции, связанные с распространением, сбором и агрегированием результатов, созданием отчета (см. рис. 19 и 20).

Рисунок 19. IDEF диаграмма взаимодействия с системой

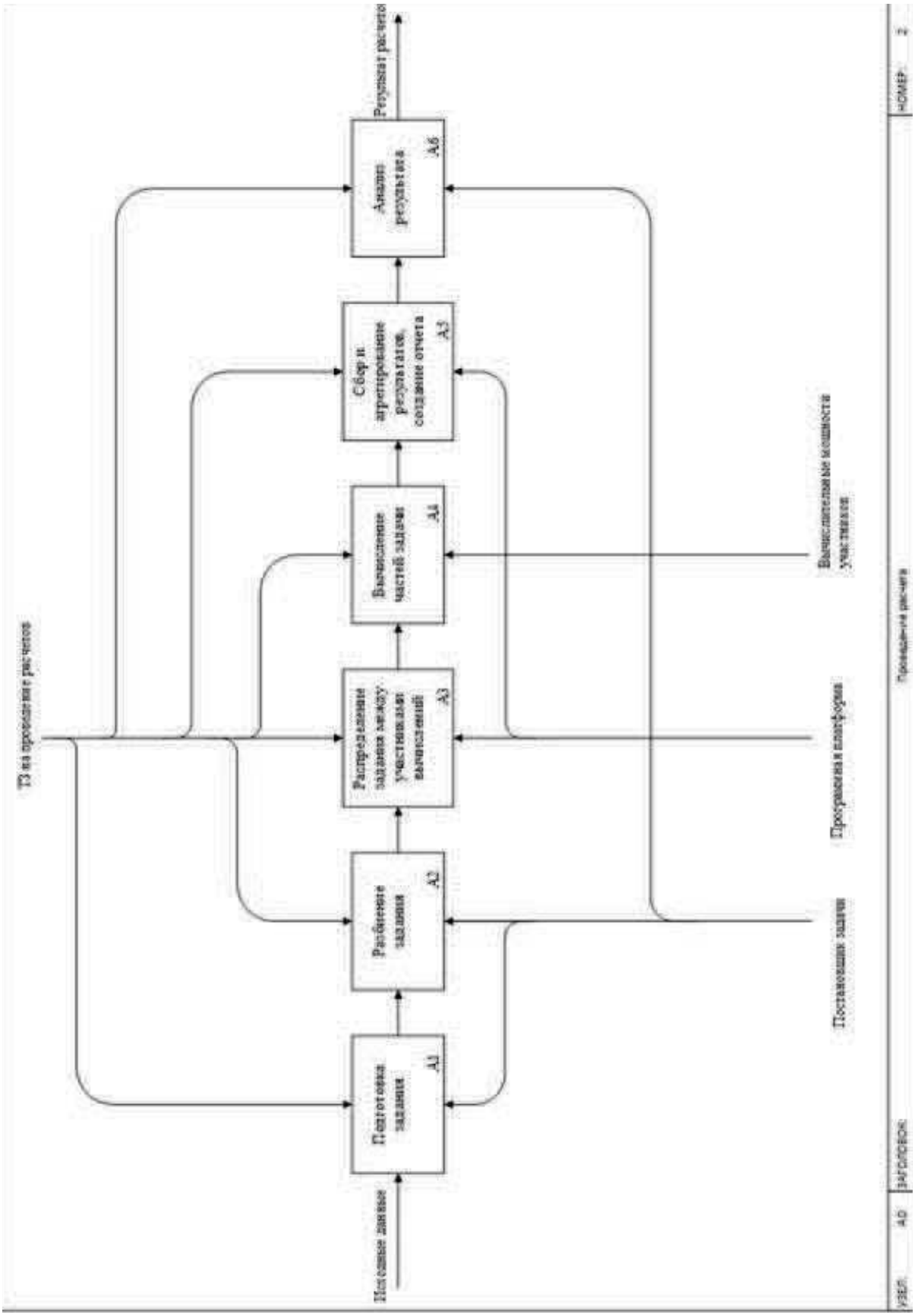


Рисунок 20. IDEF диаграмма взаимодействия с системой

ГЛАВА 5. НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ СИСТЕМЫ

5.1 Тестирование вычислительного комплекса

5.1.1 Постановка вычислительной задачи

В предыдущих главах ВКР были рассмотрены все основные темы, связанные с распределенными вычислениями в рамках модели GRID, дана общетеоретическая справка, спроектирована и развернута полноценная ГРИД среда, готовая к использованию.

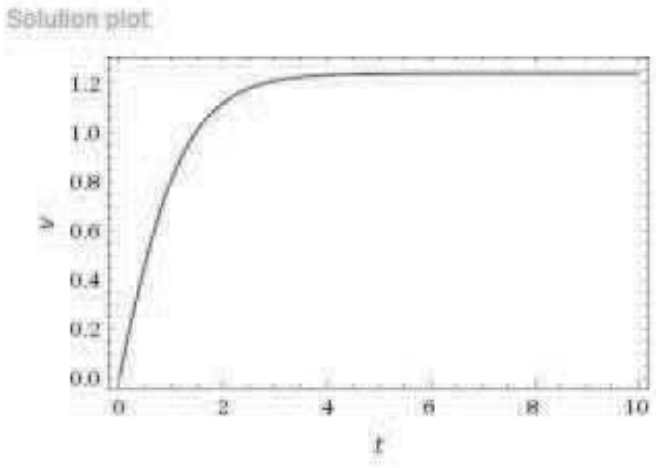


Рисунок 21 – Решение уравнения (1), полученное путем численного интегрирования в системе Wolfram Mathematica

Обратимся теперь к практической задаче. Решим задачу Коши для следующего нелинейного обыкновенного дифференциального уравнения первого порядка, при различных параметрах, входящих в ДУ. В качестве нелинейного уравнения рассмотрим следующее:

$$m * \frac{dv(t)}{dt} = -\ln(v(t) + 1) * v(t) + q * E\,, v(0) = v_0 \quad (1)$$

Преобразуем уравнение:

$$dv(t) = m^{-1}\,(-\ln(v(t) + 1) * v(t) + q * E)\,dt, v(0) = v_0 \quad (2)$$

$$v(t) = \int m^{-1}\,(-\ln(v(t) + 1) * v(t) + q * E)dt + v_0\,(3)$$

$$\forall t, v, E \in [0; +\infty)$$

$$\forall m, q \in (0; +\infty)$$

Исходное уравнение (2) описывает движение материальной точки (массивного тела массы m в пренебрежении вращательной составляющей) в вязкой среде с нелинейной зависимостью динамической тормозящей силы. Тело начинает движение с некоторой начальной скоростью v_0 в поле внешних сил напряженностью E. Параметр q отражает интенсивность силового воздействия на данное тело. В случае, к примеру, электрического поля, напряженность электрического поля и заряд, равномерно распределенный на поверхности тела.

Оценим уравнение (2) в пределе установившегося переходного процесса:

$$\lim_{\Delta t \to \infty} \left(m * \frac{dv(t)}{dt} \right) = 0 \rightarrow (-\ln(v(t) + 1) * v(t) + q * E) = 0\,(4)$$

$$(-\ln(v(t) + 1) * v(t) + q * E) = 0\,(5)$$

$$\ln(\mathrm{v(t)} + 1) * \mathrm{v(t)} = \mathrm{q} * \mathrm{E}\,(6)$$

$$\ln(\mathrm{v(t)} + 1) * \mathrm{v(t)} = \mathrm{const}\,(7)$$

Для $m,\ q,\ E = 1$ в системе Wolfram Mathematica получаем:

$$v \sim 1.23998\,(8)$$

После преобразований уравнения (2) получаем интегральное выражение (4) для скорости $v(t)$. Для интегрирования воспользуемся явным методом Эйлера.

Заменим интегральный оператор (4) конечно-разностным . Беря меньший шаг, получаем большую точность и растущее время, требуемое для завершения вычислений.

$$\boxed{v(t) = m^{-1}\left(-\ln(v(t) + 1) * v(t) + q * E\right)\Delta t + v_0 \,(9)}$$

$$\lim_{\Delta t \to 0}(9) \equiv (2)$$

Итак, параметры I = $\{v_0,\ m,\ q,\ E,\ t_l, t_r, \Delta t\}$ формируют вектор входных данных. Варьируя каждый из них, мы можем влиять на процесс вычислений, генерируя различные задания для нашей ГРИД системы и получая различные значения интеграла для разных векторов из множества.

Проверим работу численного алгоритма, скомпилированного в исполняемый файл. Получим следующие значения в выходном файле с данными:

Vector: [Velocity: 0.000000 Mass: 1.000000 Charge: 1.000000 Field: 1.000000]

-> Integral value is:1.239976

->Domain: [0.000000 ; 10.000000]

->Stepsize: 0.000025

Видно, что поле **Integral value** стремится к значению **(8)**. По данному критерию можно судить об адекватности работы скомпилированного в исполняемый модуль алгоритма. Вычисления произведены корректно.

5.1.2 Результаты тестирования

Запустим теперь данный вычислительный алгоритм в нашей системе в различных условиях.

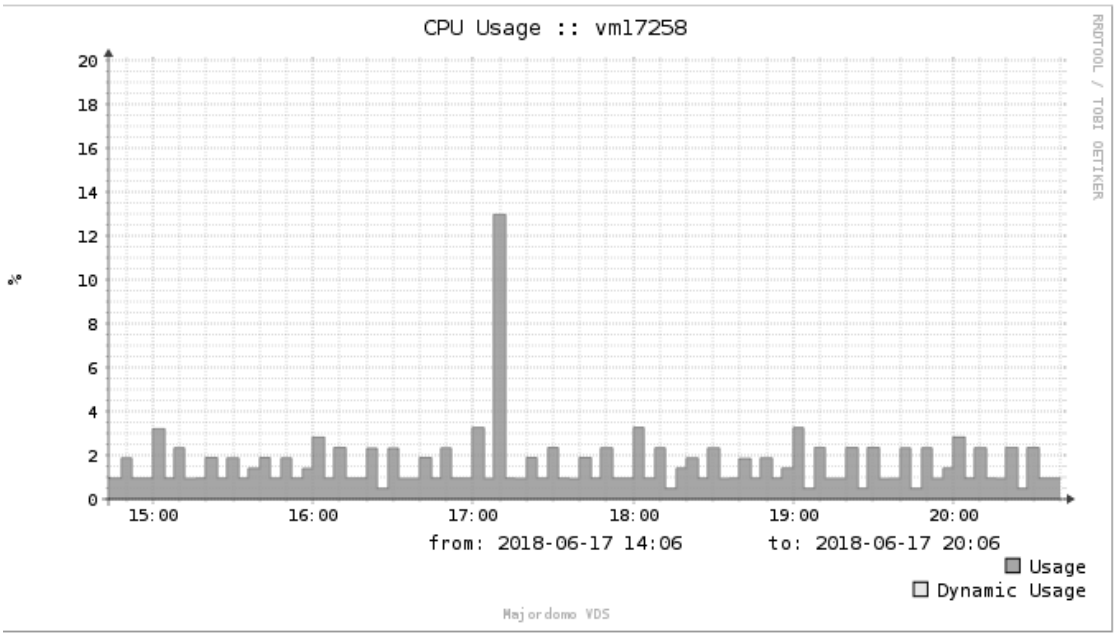
Таблица 6 - Результаты вычислений **на отрезке 0 до 10 при шаге 0.000025**

Запуск исполняемого файла на одной локальной VM машине IntelCore 2 DuoProcessorE4300 1.00 GHz:				
N	50	500	5000	20000
T	1.464	14.428	147.029	578.367
N	50000	100000	200000	-
T	-	-	-	-
Запуск исполняемого файла в рамках проекта BOINC с 1 машиной в сети: 1.Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz				
N	50	500	5000	20000
T	31.369	61.959	74.736	141.396
N	50000	100000	200000	-
T	481.966	963.932	1926.864	-
Запуск исполняемого файла в рамках проекта BOINC с BOINC **2 машинами** в сети: 1.Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz 2. Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz				
N	50	500	5000	20000
T unix (time)	-	-	-	104.971s
N	50000	100000	200000	-

T unix (time)	100.971	167.818	262.548	-
Запуск исполняемого файла в рамках проекта BOINC с BOINC **2 машинами** в сети: 1.Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz 2. Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz				
N	50	500	5000	20000
T	-	-	-	104.971s
N	50000	100000	200000	-
T	100.971	167.818	262.548	-
Запуск исполняемого файла в рамках проекта BOINC с BOINC **2 машинами** в сети: 1.Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz 2. Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz И **локального кластера** из 5 узлов IntelCore 2 DuoProcessorE4300 1.00 GHz:				
N	50	500	5000	20000
T	1.532	13.428	64.315	84.971s
N	50000	100000	200000	-
T	98.971	160.818	242.548	-

N - число входных векторов, T время, подсчитанное утилитой UNIX Time в секундах

Из таблицы 7 видно, что введение системы BOINC в качестве вычислителя позволило ускорить вычисления на большом количестве задач, однако процесс доставки результатов оказался нестабильным. Введение системы SLURM позволило сделать процесс доставки результатов детерминированным, ускорить время расчетов за счет использования ресурсов локального кластера и разгрузить сервер BOINC (Рис. 22). Пример BASH скрипта, генерирующего данные для вычислителей и исходный код приложения можно найти в приложениях Приложение A и B соответственно.



CPU Usage — использование процессора вирт. сервера в % в зависимости от тарифного плана.

Usage — % использования.

Рисунок 22 – Нагрузочная статистика использования ресурсов CPU сервера BOINC

5.2 Нагрузочное тестирование файловой системы

Тест был запущен на системе со следующей конфигурацией:

```
Volume Name: stripevol-t1
Type: Stripe
Volume ID: c733d4cb-4bf5-48be-817a-81f2d69d019b
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 5 = 5
Transport-type: tcp
Bricks:
Brick1: 192.168.20.55:/var/gfs-brick
Brick2: 192.168.20.52:/var/gfs-brick
Brick3: 192.168.20.53:/var/gfs-brick
Brick4: 192.168.20.54:/var/gfs-brick
Brick5: 192.168.20.56:/var/gfs-brick
Options Reconfigured:
transport.address-family: inet
nfs.disable: on
features.bitrot: off
```

features.scrub: Inactive

192.168.20.55:/stripevol-t1 193G 27G 167G 14% /mnt/detector1_data

5 узлов в режиме Stripe. Размер файла: 1073741824 bytes (1.1 GB)

Таблица 7 - Результаты тестирования работы файловой системы в импульсном режиме

WRITE		
1 клиент	Распределенный доступ:	Распределенный доступ (стресс-тест):
96.2688 s, 11.2 MB/s 97.3518 s, 10.2 MB/s 95.2312 s, 12.4 MB/s 96.3418 s, 11.2 MB/s	80.3751 s, 13.4 MB/s 74.4772 s, 14.4 MB/s 96.2688 s, 11.2 MB/s 97.3518 s, 10.2 MB/s	79.7394 s, 14.6 MB/s 79.1494 s, 14.6 MB/s 96.2538 s, 13.2 MB/s 96.1688 s, 13.2 MB/s
READ		
1 клиент	Распределенный доступ:	Распределенный доступ (стресс-тест):
96.3488 s, 13.2 MB/s 96.1318 s, 13.2 MB/s 96.2688 s, 13.2 MB/s 63.6136 s, 16.9 MB/s	97.5381 s, 11.0 MB/s 60.2844 s, 17.8 MB/s 97.5381 s, 11.0 MB/s 60.2845 s, 17.8 MB/s	96.1281 s, 11.1 MB/s 60.4841 s, 17.8 MB/s 97.1621 s, 11.0 MB/s 59.1242 s, 17.9 MB/s

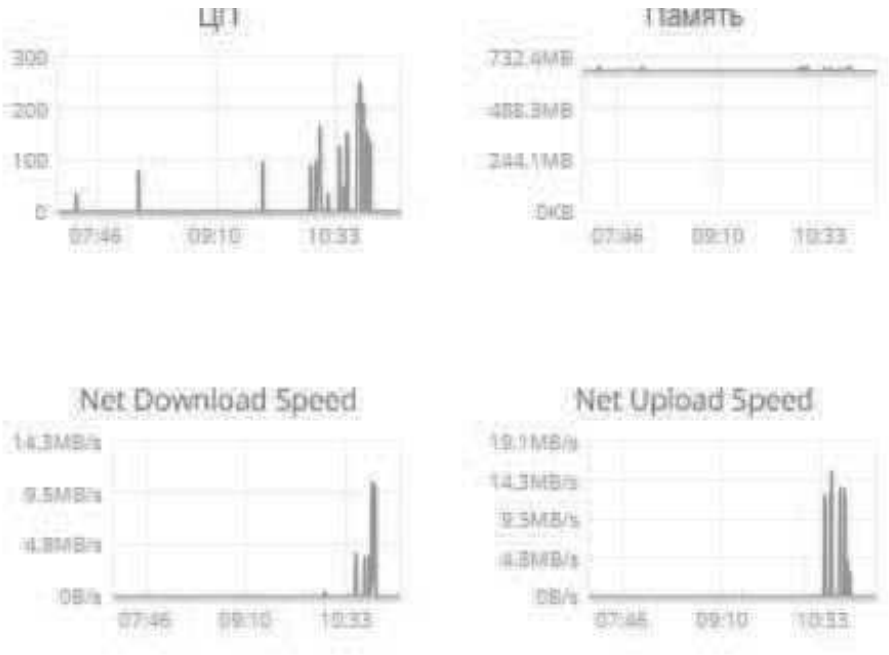


Рисунок 23. Нагрузочная статистика одного из узлов системы GlusterFS, полученная в системе OpenNebula.

Как видно из тестов, система GlusterFS позволила существенно расширить размер хранилища. Однако как только начинается передача данных, сразу же стоит учитывать скорость сетевого подключения (Рис.23, так как она может существенно снизить производительность всей файловой системы. В данном случае именно производительность сети стала ограничителем скорости передачи. Система GlusterFS в целом продемонстрировала высокую стабильность работы (uptime) в течении 2 и 4 недельных периодов тестирования.

ЗАКЛЮЧЕНИЕ

В представленной работе были разобраны все основные темы, связанные с проектированием, разработкой и вводом в эксплуатацию GRID сети для организации распределённых вычислений. Обозначены задачи, которые должны быть решены с использованием данной модели вычислений. Создана собственная GRID сеть с учетом следующих этапов развития системы.

- Планирование
- Разработка
- Внедрение
- Эксплуатация

На стадии разработки, внедрения и эксплуатации использовалось свободное ПО: BOINC, GlusterFS и SLURM. Была дана оценка текущего состояния данной системы, построены функциональные модели с полным описанием всех рабочих подсистем. Озвучены причины, по которым не производилась разработка собственного GRID ПО. Так же в ходе анализа системы BOINC в пункте 1.3 выяснилось, что она обладает рядом неприятных особенностей. В частности, их **отрицательное влияние на производительность системы** растет **пропорционально числу машин** участвующих в вычислениях и **зависит от типа выдаваемых заданий**. Так же не все задания могут быть решены только в рамках модели волонтерских вычислений, что потребовало введения введения дополнительной локальной системы обработки на базе SLURM

По итогам работы, можно делать следующий вывод:

- Получена горизонтально масштабируемая вычислительная система с минимальными затратами на организацию и последующее поддержание;
- Организовано масштабируемое файловое хранилище данных, способное хранить 10-100 ТБ данных;
- Введение системы локального планирования позволило сделать процесс доставки результатов детерминированным и ускорить время расчетов за счет использования ресурсов локального кластера.
- Научный персонал теперь может решать задачи из заданных в ТЗ классов более эффективно;
- После внедрения системы удалось автоматизировать функции, связанные с распространением, сбором и агрегированием результатов, созданием отчетных данных (см. рис. 19 и 20).

Стоить помнить, что ГРИД модель волонтерских вычислений пригодна не для всех задач и требует значительного внимания к безопасности системы (хранение данных проекта, хранение пользовательских данных). Для успешного проведения расчетов в рамках ПО BOINC, отдельные небольшие **подзадачи** должны быть очень **слабо связаны между собой** и практически **не зависеть от результатов параллельно выполняемых заданий**. В противном случае очень большие издержки производительности будут приходиться на ожидание других результатов и на их синхронизацию. SLURM частично помог решить данную проблему, введя в оперирование локальный вычислительный комплекс с предсказуемой величиной задержки. Более подробная сводка результатов опубликована в статье [41].

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. **Gordon E. Moore**. Cramming More Components onto Integrated Circuits, Electronics, 1965.
2. **Dennard, R.H., Gaensslen, F.H., Rideout, V.L., Bassous, E. and LeBlanc, A.R**.. Design of ion-implanted MOSFET's with very small physical dimensions. IEEE Journal of Solid-State Circuits, 9(5), pp.256-268, 1974.
3. **ГРИД ОИЯИ.** [Электронный ресурс] - Режим доступа: http://grid.jinr.ru/, свободный. Дата обращения: 26.03.2018.
4. **Foster, I., Kesselman, C. and Tuecke, S**. The anatomy of the grid: Enabling scalable virtual organizations. The International Journal of High Performance Computing Applications, 15(3), pp.200-222, 2001.
5. **Foster, I. and Kesselman, C**. The Grid 2: Blueprint for a new computing infrastructure. Elsevier. eds., 2003.
6. **Patterson, D.A.** Latency lags bandwith. Communications of the ACM, 47(10), pp.71-75, 2004.
7. **Bird, I..** Computing for the large hadron collider. Annual Review of Nuclear and Particle Science, 61, pp.99-118, 2011
8. **Bird, I., Carminati, F., Mount, R., Panzer-Steindel, B., Harvey, J., Fisk, I., Kersevan, B., Clarke, P., Girone, M., Buncic, P. and Cattaneo, M.** Update of the Computing Models of the WLCG and the LHC Experiments (No. CERN-LHCC-2014-014), 2014.
9. **World LHC Computing Grid** [Электронный ресурс] URL: http://wlcg.web.cern.ch/, свободный. Дата обращения: 26.03.2018.
10. **ATLAS Collaboration**. The ATLAS Experiment at the CERN Large Hadron Collider. J. Inst. 3 S08003, 2008.
11. **CMS collaboration**. The CMS experiment at the CERN LHC. J. Inst. 3, S08004, 2008.

12. **Alice Collaboration**. Performance of the ALICE Experiment at the CERN LHC. International Journal of Modern Physics A, 29(24), p.1430044, 2014.
13. **Научно-методический семинар ЛФВЭ №9 -"The new Inner Tracking System of the ALICE experiment",** [Электронный ресурс] - Режим доступа: http://indico.jinr.ru/conferenceDisplay.py?confId=289, свободный. Дата обращения: 26.03.2018.
14. **Markram, H., Meier, K., Lippert, T., Grillner, S., Frackowiak, R., Dehaene, S., Knoll, A., Sompolinsky, H., Verstreken, K., DeFelipe, J. and Grant, S.** Introducing the human brain project. Procedia Computer Science, 7, pp.39-42, 2011.
15. **The Human Brain Project.** [Электронный ресурс] - Режим доступа: https://www.humanbrainproject.eu/, свободный. Дата обращения: 26.03.2018.
16. **Markram, H., Muller, E., Ramaswamy, S., Reimann, M.W., Abdellah, M., Sanchez, C.A., Ailamaki, A., Alonso-Nanclares, L., Antille, N., Arsever, S. and Kahou, G.A.A**. Reconstruction and simulation of neocortical microcircuitry. Cell, 163(2), pp.456-492, 2015.
17. **Amunts, K., Ebell, C., Muller, J., Telefont, M., Knoll, A. and Lippert, T.** The human brain project: creating a European research infrastructure to decode the human brain. Neuron, 92(3), pp.574-581, 2016.
18. **CSCS - Swiss National Supercomputing Centre** [Электронный ресурс] - Режим доступа: http://www.cscs.ch/, свободный. Дата обращения: 26.03.2018.
19. **SP7-HPAC HBP** [Электронный ресурс] - Режим доступа: https://www.humanbrainproject.eu/en/about/project-structure/subprojects/#SP7, свободный. Дата обращения: 26.03.2018.
20. **High Performance Analytics & Computing Platform** [Электронный ресурс] - Режим доступа: https://hbp-hpc-platform.fz-juelich.de/, свободный. Дата обращения: 26.03.2018.

21. **Data Acquisition Systems (DAQ) and Equipment** [Электронный ресурс] - Режим доступа: https://www.industrial-electronics.com/DAQ/, свободный. Дата обращения: 26.03.2018.
22. **Fahringer, T., Prodan, R., Duan, R., Hofer, J., Nadeem, F., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.L. and Villazon, A.** Askalon: A development and grid computing environment for scientific workflows. Workflows for e-Science, pp. 450-471, 2007.
23. **Blomer, J.** Experiences on File Systems: Which is the best file system for you? Journal of Physics: Conference Series. Vol. 664. No. 4. IOP Publishing, 2015.
24. **Benjamin Depardon, Gaël Le Mahec, and Cyril Séguin.** Analysis of six distributed file systems. Research Report. The open archive HAL, 2013.
25. **Blomer, J., Buncic, P., Charalampidis, I., Harutyunyan, A., Larsen, D. and Meusel, R.** Status and future perspectives of CernVM-FS. In Journal of Physics: Conference Series (Vol. 396, No. 5, p. 052013). IOP Publishing, 2012.
26. **Shvachko, K., Kuang, H., Radia, S. and Chansler, R.**. The hadoop distributed file system. In Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on (pp. 1-10). IEEE, 2010.
27. **Schwan, P.** Lustre: Building a file system for 1000-node clusters. In Proceedings of the 2003 Linux symposium. Vol. 2003, pp. 380-386, 2003.
28. **Nagle, D., Serenyi, D. and Matthews, A.** The panasas activescale storage cluster: Delivering scalable high bandwidth storage. In Proceedings of the 2004 ACM/IEEE conference on Supercomputing, p. 53, IEEE Computer Society, 2004.
29. **Weil S. A..** Ceph: reliable, scalable, and high-performance distributed storage Ph.D. thesis University of California Santa Cruz, 2007.
30. **Gluster Inc.** An Introduction to Gluster Architecture. Whitepaper, 2011.

31. **Fuhrmann, P. and Gülzow, V.** dCache, storage system for the future. In European Conference on Parallel Processing, pp. 1106-1113, 2006.
32. **GlusterFS Documentation** [Электронный ресурс] - Режим доступа: https://docs.gluster.org/, свободный. Дата обращения: 26.03.2018.
33. **Reuther, A., Byun, C., Arcand, W., Bestor, D., Bergeron, B., Hubbell, M., Jones, M., Michaleas, P., Prout, A., Rosa, A. and Kepner, J.** Scalable system scheduling for HPC and big data. Journal of Parallel and Distributed Computing, 111, pp.76-92, 2018.
34. **The Linux man-pages project** [Электронный ресурс] - Режим доступа: https://www.kernel.org/doc/man-pages/, свободный. Дата обращения: 26.03.2018.
35. **Batch Systems Comparison** [Электронный ресурс] - Режим доступа: https://twiki.cern.ch/twiki/bin/view/LCG/BatchSystemComparison/, свободный. Дата обращения: 26.03.2018.
36. **Andy B. Yoo, Morris A. Jette, and Mark Grondona**. Slurm: Simple linux utility for resource management. Workshop on Job Scheduling Strategies for Parallel Processing, 2003.
37. **SLURM Documentation** [Электронный ресурс] - Режим доступа: https://slurm.schedmd.com/, свободный. Дата обращения: 26.03.2018.
38. **David P. Anderson** Boinc: A system for public-resource computing and storage. Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing. IEEE Computer Society, 2004.
39. **Computing with BOINC.** [Электронный ресурс] -Режим доступа: https://boinc.berkeley.edu/trac/wiki/ProjectMain, свободный. Дата обращения: 22.03.2018.
40. **Официальный сайт LHC@home.** [Электронный ресурс] -Режим доступа: http://lhcathome.web.cern.ch/, свободный. Дата обращения: 22.03.2018.

41. **Kashansky, V.V. and Kaftannikov, I.L.** Application of SLURM, BOINC, and GlusterFS as Software System for Sustainable Modeling and Data Analytics. EPJ Web of Conferences, Vol. 173, p. 05010, 2018.

ПРИЛОЖЕНИЕ А

Исходный код исполняемого модуля для системы BOINC на C/C++

```
// This file is part of BOINC.
// http://boinc.berkeley.edu
// Copyright (C) 2008 University of California
//
// BOINC is free software; you can redistribute it and/or modify it
// under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation,
// either version 3 of the License, or (at your option) any later version.
//
// BOINC is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
// See the GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with BOINC.  If not, see <http://www.gnu.org/licenses/>.

// This program serves as both
// - An example BOINC application, illustrating the use of the BOINC API
// - A program for testing various features of BOINC
//
// NOTE: this file exists as both
// boinc/apps/upper_case.cpp
// and
// boinc_samples/example_app/uc2.cpp
// If you update one, please update the other!

// The program converts a mixed-case file to upper case:
// read "in", convert to upper case, write to "out"
//
// command line options
// --cpu_time N: use about N CPU seconds after copying files
// --critical_section: run most of the time in a critical section
// --early_exit: exit(10) after 30 chars
// --early_crash: crash after 30 chars
// --run_slow: sleep 1 second after each character
// --trickle_up: sent a trickle-up message
// --trickle_down: receive a trickle-up message
// --network_usage: tell the client we used some network
//

#include "uc2.h"

double euler_solve_equation( param_vector *vec ) {

       // explicit Euler's method
        // Solves m * dv/dt = -ln(v)*v + (q*E), non-linear medium with logaritm speed
drag-force

        // regime settings
       double v = vec->velocity, m=vec->mass, q=vec->charge, E=vec->E_field, t1=vec->lb,
t2=vec->rb, stepsize=vec->ssz;

       // integrating
       for(t1;t1<=t2;t1+=stepsize) {
              v+= ((1/m)*((-log(v+1)*v) + ((q/m)*E)))*stepsize;
                //v+= ((1/m)*(1-v))*stepsize;
```

```
       

       // integral value
       return v;
}

int do_checkpoint(MFILE mf, int nrows) {

    int retval;
    string resolved_name;

    FILE* f = fopen("temp", "w");

    if (!f) return 1;

    // writes count of processed vector rows
       fprintf(f, "d", nrows);
    fclose(f);

    retval = mf.flush();
    if (retval) return retval;

       boinc_resolve_filename_s(CHECPOINT_FILE, resolved_name);

    // saved to CHECPOINT_FILE name
       retval = boinc_rename("temp", resolved_name.c_str());

       if (retval) return retval;

    return 0;
}

static double do_some_computing(int foo) {
    double x = 3.11*foo;
    int i;
    for (i=0; i<0000000; i++) {
        x += .12313123;
        x *= 0.383;
    }
    return x;
}

int main(int argc, char **argv) {

    double integral_value=0;

    param_vector    ival;

                    ival.velocity = 0;//1.1
                    ival.mass=1;
                    ival.charge=2;
                    ival.E_field=0.;
                                ival.lb=0;
                                ival.rb=3;
                                ival.ssz=0.02;
       /*
        *     Inital values
        *
        *
        *
        */

    int i, fsize;
    int c, nrows = 0, retval, n=0;
    double fd;
```

```
   checkpoint_vector    chckp_vec;
                                       chckp_vec.cur_row_dmp_pv = ival;
                                       chckp_vec.cur_row_integral_value = 0;
                                       chckp_vec.cur_row_time_moment = 0;
                                       chckp_vec.rows_processed = 0;

char input_path□□12□, output_path□□12□, chkpt_path□□12□, buf□2□□□;


MFILE out;
FILE* state, *infile;

for (i=0; i<argc; i++) {

    if (strstr(argv□i□, "early_exit")) early_exit = true;
    if (strstr(argv□i□, "early_crash")) early_crash = true;
    if (strstr(argv□i□, "early_sleep")) early_sleep = true;
    if (strstr(argv□i□, "run_slow")) run_slow = true;
    if (strstr(argv□i□, "critical_section")) critical_section = true;
    if (strstr(argv□i□, "network_usage")) network_usage = true;

    if (strstr(argv□i□, "cpu_time")) {
        cpu_time = atof(argv□++i□);
    □
    if (strstr(argv□i□, "trickle_up")) trickle_up = true;
    if (strstr(argv□i□, "trickle_down")) trickle_down = true;
□ // O□

retval = boinc_init();

if (retval) {
    fprintf(stderr, "□s boinc_init returned □d□n",
        boinc_msg_prefix(buf, sizeof(buf)), retval
    );
    exit(retval);
□ // O□

fprintf(stderr, "□s app started; CPU time □f, flags:□s□s□s□s□s□s□s□n",
    boinc_msg_prefix(buf, sizeof(buf)),
    cpu_time,
    early_exit□" early_exit":"",
    early_crash□" early_crash":"",
    early_sleep□" early_sleep":"",
    run_slow□" run_slow":"",
    critical_section□" critical_section":"",
    trickle_up□" trickle_up":"",
    trickle_down□" trickle_down":""
); // O□

// open the input file (resolve logical name first)
//
boinc_resolve_filename(INPUT_FILENAME, input_path, sizeof(input_path));

infile = boinc_fopen(input_path, "r");

   if (!infile) {
    fprintf(stderr,
        "□s Couldn't find input file, resolved name □s.□n",
        boinc_msg_prefix(buf, sizeof(buf)), input_path
    );
    exit(-1);
□

// get size of input file (used to compute fraction done)
```

```
    //
    //file_size(input_path, fsize);

       //save entry lines count to fsize
       //for(fsize=0; fscanf(infile, "□lf □lf □lf □lf □lf □lf □lf",□ival.velocity,
□ival.mass, □ival.charge, □ival.E_field, □ival.lb, □ival.rb, □ival.ssz) != EOF; fsize++);
       //fseek(infile, 0, SEE□_SET);

    boinc_resolve_filename(OUTPUT_FILENAME, output_path, sizeof(output_path));

    // See if there's a valid checkpoint file.
    // If so seek input file and truncate output file
    //
    boinc_resolve_filename(CHEC□POINT_FILE, chkpt_path, sizeof(chkpt_path));

    state = boinc_fopen(chkpt_path, "r");

    if (state) {
        n = fscanf(state, "□d", □nrows);
        fclose(state);
    □

       //fprintf(stderr, "□d lines□n", nrows);

       //exit(-1);

    if (state □□ n) {
             // dont process CHEC□POINT_FILE
        //fseek(infile, nrows, SEE□_SET);
        //boinc_truncate(output_path, nrows);
        retval = out.open(output_path, "ab");

       □ else {
        retval = out.open(output_path, "wb");
    □

    if (retval) {

        fprintf(stderr, "□s APP: non-lin_ode_analysis output open failed:□n",
            boinc_msg_prefix(buf, sizeof(buf))
        );

        fprintf(stderr, "□s resolved name □s, retval □d□n",
            boinc_msg_prefix(buf, sizeof(buf)), output_path, retval
        );

        perror("open");

        exit(1);
    □

    if (network_usage) {
        boinc_network_usage(□., 1□.);
    □

    // main loop - read characters, convert to UC, write
    // □ entries nrows =3

    for(int i=0; i<nrows □□ nrows; i++) {
             if(fscanf(infile, "□lf □lf □lf □lf □lf □lf □lf",□ival.velocity, □ival.mass,
□ival.charge, □ival.E_field, □ival.lb, □ival.rb, □ival.ssz) == EOF) {
                   perror("Error: End of file reached!");
                   exit(-1);
             □
       □
```

```
    for(int i=nrows; fscanf(infile, "□lf □lf □lf □lf □lf □lf □lf",□ival.velocity,
□ival.mass, □ival.charge, □ival.E_field, □ival.lb, □ival.rb, □ival.ssz) != EOF; i++) {

              // ival is read, create thread here ...
              integral_value = euler_solve_equation( □ival );

              // save integral value, row is processed
              out.printf("□□ector: □□lf □lf □lf □lf□ □□lf □lf □lf□ -> Integral value is:
□lf□□n", ival.velocity, ival.mass, ival.charge, ival.E_field,ival.lb, ival.rb, ival.ssz,
integral_value);
        nrows++;

        if (run_slow) {
            boinc_sleep(1.);
        □

        if (early_exit □□ i>30) {
            exit(-10);
        □

        if (early_crash □□ i>30) {
            boinc_crash();
        □
        if (early_sleep □□ i>30) {
            boinc_disable_timer_thread = true;
            while (1) boinc_sleep(1);
        □

        if (boinc_time_to_checkpoint()) {

                    retval = do_checkpoint(out, nrows);

            if (retval) {

                fprintf(stderr, "□s APP: non-lin_ode_analysis checkpoint failed □d□n",
                    boinc_msg_prefix(buf, sizeof(buf)), retval
                );

                exit(retval);

            □

            boinc_checkpoint_completed();

        □

              if (report_fraction_done) {

                    //fd = nrows/fsize;

                    //if (cpu_time) fd /= 2;

                    //boinc_fraction_done(fd);

              □
    □

    retval = out.flush();

    if (retval) {

              fprintf(stderr, "□s APP: non-lin_ode_analysis flush failed □d□n",
            boinc_msg_prefix(buf, sizeof(buf)), retval
        );
```

```
        exit(1);
    }

    if (trickle_up) {

        boinc_send_trickle_up(
            const_cast<char*>("non-lin_ode_analysis"),
            const_cast<char*>("sample trickle message")
        );

    }

    if (trickle_down) {

        boinc_sleep(10);

             retval = boinc_receive_trickle_down(buf, sizeof(buf));

             if (!retval) {
            fprintf(stderr, "Got trickle-down message: %s\n", buf);
        }

    }

    boinc_fraction_done(1);

    boinc_finish(0);
}
```

ПРИЛОЖЕНИЕ В

Исходный код BASH скрипта (дебаг вариант)

```
#!/bin/bash

# Generate 20 files with 10 rows

nflies = 2
nrows = 10

for ((i = 0; i<nflies; i++)); do

       filename = "dat_"(date + F_H_M_S)"_micst_"nrows"ent_perdata_"i

       echo - e i": Next "nrows" rows ... n"
       for (( = 0; <nrows; ++)); do

              echo - e (shuf - i 0 - 2 - n 1)"."(shuf - i 0 - - n 1)" 1.000 1.000
1.000 0.000 3.000 0.00002nn" >> . / workunits / filename
              done
              done

              for fname in (ls ./ workunits); do

                     cp ./ workunits / fname ./ bin / dir_hier_path fname
                     #rm - f ./ workunits / fname
# --target_user 8

                     ./bin/create_work --appname non - lin_ode_analysis --min_quorum 1 --
target_nresults 1 --wu_name wu_nloa_fname --wu_template templates/default_in --
result_template templates/default_out fname

                     done

                     rm - f . / workunits/*
```

ПРИЛОЖЕНИЕ С

Конфигурационный файл SLURM

```
# slurm.conf file generated by configurator.html.
# Put this file on all nodes of your cluster.
# See the slurm.conf man page for more information.
#
ControlMachine=linux-cl
#
AuthType=auth/none
CacheGroups=0
CryptoType=crypto/openssl
Mpi□efault=none
ProctrackType=proctrack/pgid
ReturnToService=1
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmctldPort=□81□
SlurmdPidFile=/var/run/slurmd.pid
SlurmdPort=□818
SlurmdSpool□ir=/tmp/slurmd
SlurmUser=slurm
StateSaveLocation=/tmp
SwitchType=switch/none
TaskPlugin=task/none
#
# TIMERS
InactiveLimit=0
□illWait=30
Min□obAge=300
SlurmctldTimeout=120
SlurmdTimeout=300
Waittime=0
#
# SCHE□ULING
FastSchedule=1
SchedulerType=sched/backfill
SchedulerPort=□321
SelectType=select/linear
#
# LOGGING AN□ ACCOUNTING
AccountingStorageType=accounting_storage/none
ClusterName=cluster
□obCompType=□obcomp/none
□obCredentialPrivate□ey = /usr/local/etc/slurm.key
□obCredentialPublicCertificate = /usr/local/etc/slurm.cert
□obAcctGatherFrequency=30
□obAcctGatherType=□obacct_gather/none
Slurmctld□ebug=3
Slurmd□ebug=3
#
# COMPUTE NO□ES
NodeName=linux□1-32□ CPUs=1 State=UN□NOWN
PartitionName=debug Nodes=linux□1-32□ □efault=YES MaxTime=INFINITE State=UP
```