Федеральное государственное учреждение «Федеральный научный центр
Научно-исследовательский институт системных исследований
Российской академии наук»
(ФГУ ФНЦ НИИСИ РАН)

ТРУДЫ НИИСИ РАН

ТОМ 11 № 4

МАТЕМАТИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ:

ТЕОРЕТИЧЕСКИЕ И ПРИКЛАДНЫЕ АСПЕКТЫ

МОСКВА
2021

Тематика номера:
Информационные и компьютерные технологии, развитие суперкомпьютерных технологий и оптимизация суперкомпьютерных вычислений, математическое моделирование и программное обеспечение технологических процессов, проектирование и моделирование СБИС, математические исследования, информационные технологии в учебной информатике

Журнал публикует оригинальные статьи по следующим областям исследований: математика, математическое и компьютерное моделирование, обработка изображений, визуализация, системный анализ, методы обработки сигналов, информационная безопасность, информационные технологии, высокопроизводительные вычисления, оптико-нейронные технологии, микро- и наноэлектроника, вопросы численного анализа, история науки и техники.

The topic of the issue:

Information and computer technologies, development of supercomputer technologies and optimization of supercomputer computations, mathematical modeling and software of technological processes, design and modeling of VLSI, mathematical issues, information technology in educational informatics

The Journal publishes novel articles on the following research arias: mathematics, mathematical and computer modeling, image processing, visualization, system analysis, signal processing, information security, information technologies, high-performance computing, optical-neural technologies, micro- and nanoelectronics, problems of numerical analysis, history of science and of technique.

Заведующий редакцией: В.Е. Текунов

Издатель: ФГУ ФНЦ НИИСИ РАН,
117218, Москва, Нахимовский проспект 36, к. 1

СОДЕРЖАНИЕ

CONTENTS

Современные распределенные системы хранения данных: оценка применимости и методики тестирования

А.Б. Бетелин[1], И.Б. Егорычев[2], А.А. Прилипко[3], Г.А. Прилипко[4], С.Г. Романюк[5], Д.В. Самборский[6]

[1]ab@niisi.msk.ru, [2]egorychev@niisi.msk.ru, [3]aaprilipko@niisi.msk.ru, [4]prilipko@niisi.msk.ru, [5]sgrom@niisi.ras.ru, [6]samborsky_d@fastmail.com

ФГУ ФНЦ НИИСИ РАН, Москва, Россия

Аннотация. Данная статья содержит обзор современных распределенных систем хранения данных (СХД) с открытым исходным кодом. Популярность масштабируемых СХД связана с растущей потребностью хранения и анализа больших объемов данных. Проблема выбора конкретной системы осложняется разнообразием использованных программных архитектур, и соответствующих им функциональных характеристик. В статье анализируются основные факторы, определяющие соответствие СХД требованиям пользователей и прикладных программ, и предлагается методология тестирования и развертывания выбранной системы.

Ключевые слова: распределенная система хранения данных, файловая система, горизонтальное масштабирование

1. Введение. История архитектур систем хранения данных

Исторически централизованные системы хранения данных (СХД) были созданы для крупных серверов, обеспечивающих работу многих пользователей и программных сервисов. Хранилища данных как отдельный сервис понадобились при переходе к клиент-серверным технологиям и использованию локальных сетей, соединяющих персональные компьютеры, рабочие станции и серверы. При этом получили распространение два вида доступа к данным: сетевые файл-серверы (Network-Attached Storage, NAS), предоставляющие доступ к файловой системе хранилища данных по специальному протоколу передачи данных (NFS, SMB, FTP), и сети хранения данных (Storage Area Network, SAN), в которых доступ к физическим и логическим блочным устройствам предоставляется по каналам связи Fiber Channel, Ethernet или InfiniBand.

Одновременно с развитием этих технологий был разработан POSIX-стандарт интерфейса операционных Unix-подобных систем, который описывает набор операций с файлами и каталогами. Оказалось, что требования полной поддержки POSIX-стандарта [38] практически невыполнимы для сетевых файловых систем, поэтому NAS-хранилища не всегда могут быть использованы с прикладными программами, созданными для работы с локальными файловыми системами. В качестве компромисса часто применяется смешанная стратегия, в которой NAS-хранилища используются службами, не требующими скоростного доступа (такими как сетевые файл-серверы или хранилища архивных данных), а SAN-ресурсы передаются в монопольное использование тем программам, которым необходима POSIX-совместимая файловая система или которые предъявляют повышенные требования к скорости ввода-вывода (примерами служат серверы управления базами данных и системы виртуализации). Для обеспечения отказоустойчивости вычислительные узлы объединяются в кластер высокой доступности (High Availability cluster, HA-кластер) и подключаются к системе хранения данных по выделенной локальной сети с резервированием каналов. В таком случае при отказе оборудования одного узла, его функции автоматически передаются другому узлу с максимальным периодом недоступности около одной минуты.

Дальнейшее развитие систем хранения данных выразилось в появлении следующих новых технологий: кластерных файловых систем, распределенных файловых систем, специализированных СХД с высокой степенью масштабируемости и надежности.

Кластерные файловые системы обеспечивают полноценный интерфейс разделяемой файловой системы на всех узлах вычислительного кластера. В качестве устройств хранения данных кластерные файловые системы (ФС) используют разделяемые SAN-устройства, а координацию

доступа к файлам и каталогам регулирует драйвер файловой системы вместе с распределенным менеджером блокировок. Работа кластерной файловой системы с поддержкой всех описанных стандартом POSIX операций требует координации доступа к файлам и каталогам и синхронизации файловых кэшей на всех узлах вычислительного кластера.

В любых распределенных системах хранения данных и системах управления оказывается невозможно гарантировать одновременно их консистентность и доступность в условиях временных потерь связности кластера. Это условие впервые было сформулировано в известной CAP-теореме [1] по отношению к системам управления распределенными базами данных (СУБД). Поэтому на практике приходится идти на компромисс, жертвуя одной из этих трех характеристик в пользу двух других. В случае кластерных ФС, как и для СУБД, приоритетна консистентность – при отказе устройств и потери связности кластера останавливается работа всей файловой системы. Но и в случае штатной работы необходимость поддержки семантики всех файловых операций вынуждает кластерную ФС устанавливать и синхронизировать блокировки почти для каждой единичной операции с файлами и каталогами.

По этим причинам производительность кластерных ФС резко падает с увеличением количества узлов в кластере, и, фактически, одну файловую систему можно использовать не более чем на нескольких десятках узлов. Вышеописанные недостатки кластерных ФС означают, что эти системы не могут быть причислены к классу распределенных СХД.

1.1. Соотношение скоростей устройств ввода-вывода

С момента распространения клиент-серверных технологий скорость локальных сетей по отношению к скорости работы накопителей стала ключевой характеристикой, определяющей построение систем хранения данных. За последние 30 лет соотношение этих скоростей в массовом оборудовании несколько раз менялось, см. Таблицу 1.

В период 1990-2000 гг., когда сеть была существенно медленнее дисковых массивов, получили распространение только файл-серверы для централизованного хранения больших объемов данных (так называемые «холодные данные»). Затем, с появлением оптоволоконных высокоскоростных каналов Fiber Channel и Ethernet 1Gbit, оказалось возможным подключать дисковые массивы без существенной потери скорости обмена и увеличения задержек. Это дало толчок развитию SAN-сетей, хотя соответствующее оборудование было еще достаточно дорогим. С появлением твердотельных накопителей (SAS/SATA SSD, NVMe SSD) скорость произвольного доступа к данным увеличилась на несколько порядков, и сетевые соединения снова стали ограничивать скорость СХД. Позже на рынке появилось относительно недорогое оборудование для высокоскоростных сетей InfiniBand, ранее используемое только в суперкомпьютерных центрах для параллельных вычислений. Сеть InfiniBand предусматривает аппаратную поддержку прямых пересылок областей памяти (Remote Direct Memory Access, RDMA), которая снижает нагрузку на центральный процессор сервера и разрешает пересылку данных без лишних копирований и переключений в режим ядра ОС и обратно. На базе Ethernet также можно использовать RDMA (RDMA over converged Ethernet, RoCE), если такой режим работы поддержан оборудованием и системным ПО. Пересылки данных по протоколу RDMA используются для уменьшения задержек в каналах, связывающих СХД с потребителями.

Таблица 1. Сравнение скоростей нескольких поколений сетевого оборудования, и накопителей данных, 1990-2020 гг.

Годы	Сетевая технология	Скорость (bandwidth), Мб/сек	Технология накопителей данных	Скорость (bandwidth), Мб/сек
1990-2000	Ethernet 10 Mbit	1	HDD	0.5-50
1995-2005	Ethernet 100 Mbit	10	HDD	10-100
2000-2015	FC 1-10GFP	100-1200	HDD+SSD SAS/SATA	100-1000
2000-2010	Ethernet 1 Gbit	100	SSD SAS/SATA, NVMe	200-3000
2010-2020	Ethernet 10 Gbit	1000		
2015-2020	Fiber Channel 32GFC	3200		
2015-2020	Ethernet 40, 100 Gbit	4000-10000		
2015-2020	InfiniBand	4000-6000		

Чтобы использовать возможности новых сетевых технологий для подключения накопителей данных было предложено расширить спецификацию NVMe и позволить передавать пакеты данных за пределы шины PCI Express. Эта технология получила название NVMe over Fabric (NVMe-oF), она позволяет использовать каналы Fiber Channel или RDMA-пересылки на базе Infiniband или RoCE. Технология NVMe-oF, подобно iSCSI, способна предоставить доступ к NVMe-накопителям через коммутируемую локальную сеть, что позволяет создавать так называемые all-flash-array (AFA) SAN-ресурсы, обеспечивающие недостижимые ранее скорости в миллионы операций в секунду (IOPS).

2. Распределенные системы хранения данных

Почти одновременно с распространением кластерных файловых систем разрабатывались идеи построения распределенных отказоустойчивых масштабируемых (и параллельных) файловых систем и хранилищ объектов данных. Стимулами для развития этих систем послужила необходимость горизонтального масштабирования СХД и увеличение скорости локальных сетей по сравнению со скоростью накопителей данных.

При создании распределенных ФС обычно декларируются следующие принципы:

- разделение служб хранения данных, метаданных, и служб диспетчеров обслуживания клиентов;
- прямая связь клиента ФС с узлами хранения данных после успешного открытия доступа к файлу или объекту (с использованием выделенной скоростной сети для передачи данных data plane);
- дупликация при хранении и контроль целостности данных при чтении за счет добавления кодов коррекции ошибок;
- самовосстановление после сбоя и частичной потери связности узлов сети;
- частичный или полный отказ от семантики файловых операций стандарта POSIX;
- теоретически неограниченное горизонтальное масштабирование по числу серверов хранения данных и числу клиентов;
- кластерах). возможность параллельного потокового обмена данными одновременно со многими узлами хранения (эта функция необходима для работы параллельных программ на вычислительных

Не все эти подходы были обязательно использованы при создании каждой конкретной ФС, разные системы определяют свои цели и приоритеты.

Основная идея распределенных СХД состоит в том, что в крупных хранилищах данных настоящие коллизии одновременного доступа к одним и тем же файлам или объектам либо полностью исключены, либо относительно редки, и поэтому обслуживание клиентов можно сделать параллельным, при условии, что система хранения данных не будет тесно связана с устройством организации доступа к ним. Действительно, в кластерных ФС система синхронизации доступа к файлам и каталогам сложна и избыточна, поскольку традиционное устройство дисковых файловых систем вынуждает получать блокировку монопольного доступа к устройству хранения данных даже для логически независимых операций (например, запись в разные файлы или создание разных файлов в общем каталоге).

К категории распределенных СХД можно условно причислить и упрощенные масштабируемые специализированные хранилища. Например, простейшей системой хранения является распределенное хранилище пар ключ-значение (key-value), для которого ограничен набор операций с данными и допускается отложенная консистентность данных (‘eventual consistency’). Размер такого хранилища можно наращивать практически неограниченно без потерь производительности. На другой стороне спектра сложности распределенных СХД находятся системы, которые на базе хранилища объектов конструируют настоящие файловые системы с частичной поддержкой семантики файловых операций стандарта POSIX (примерами служат CephFS и GlusterFS). Кроме этого разделения, распределенные СХД еще различаются по следующим характеристикам:

- по уровню отказоустойчивости (образуют ли службы кластер высокой доступности);
- поддержкой избыточного хранения данных методом репликации или более эффективным кодированием;
- гранулярностью степени репликации (файл, папка, том);
- поддержкой расслоения данных файлов для ускорения параллельных программ (sharding или striping);
- методом подключения клиентов – с помощью модулей ядра ОС, модулями FUSE, сетевыми протоколами HTTP(s), SFTP, SMB, NFS, либо специальными библиотеками для прикладных программ.

Сложность таких систем часто приводит к тому, что ради частичного упрощения разработчики идут на компромисс и жертвуют одной характеристикой ради другой (в случае Ceph, это надежность и масштабируемость в ущерб скорости и доступности). Можно предположить, что в силу теоретических и практических ограничений не существует решения для задачи построения масштабируемой распределенной СХД, которое бы одновременно удовлетворяло всем требованиям.

Тем не менее такие распределенные СХД становятся все более популярны, а практика объединения функций хранения и обработки данных на узлах вычислительного кластера в среде маркетинга получила название «гиперконвергентной инфраструктуры» (HCI). Предполагалось, что этот подход мог бы упростить и удешевить построение крупных вычислительных кластеров. Например, можно было бы запускать виртуальные ОС и хранить их диски на одинаково настроенных узлах, сохранив при этом возможность центрального управления и миграции виртуальных ОС для балансировки нагрузки кластера. Однако на практике пока не удается реализовать такую архитектуру без существенных потерь производительности и надежности (например, см. описание опыта использования Ceph RBD у web-хостера FirstVDS [39]).

На данный момент ни одна из распределенных СХД с открытым исходным кодом не может в полной мере обеспечить локальность хранения данных. Имеется в виду хранение одной из копий данных ресурса (файла, объекта, или виртуального диска) на том же узле кластера, где расположены его потребители – виртуальные ОС или прикладные системы. Такое расположение ускорило бы чтение, и, возможно, запись данных, если допускается подтверждение факта записи до завершения синхронизации состояния этого ресурса с его другими копиями. Наиболее полно этой модели соответствует СХД Linstor, в которой есть полуавтоматическое управление расположением томов сетевых дисков DRBD. Хотя система Linstor не является распределенной файловой системой, тем не менее она часто используется в качестве хранилища для сред виртуализации и контейнеризации.

Значительная потеря производительности в работе распределенных ФС по сравнению с локальными накопителями данных побуждает разработчиков применять различные алгоритмы кэширования. Например, в версии 2.13 файловой системы Lustre была добавлена функция полуавтоматического кэширования данных на стороне клиента, Persistent Client Cache (PCC) [7]. В качестве ресурса для кэширования предлагается использовать скоростные NVMe SSD накопители или энергонезависимые модули памяти PMEM. Кэширование допускается либо в режиме чтения (read-only, RO-PCC), либо в режиме отложенной записи (writeback, RW-PCC). При этом в режиме RO-PCC запись в файл запрещается также для всех остальных клиентов системы, что позволяет использовать блокировку записи вместо алгоритма синхронизации кэшей. Режим RW-PCC предназначен для монопольного доступа к файлу одним клиентом, он запрещает запись в файл остальным клиентам и не гарантирует для них своевременного обновления измененных данных, что также упрощает алгоритм работы файловой системы. В целом, функция локального кэширования Lustre состоит в ускорении прикладных программ, использующих файловый ввод-вывод в одном из двух режимов: либо в режиме чтения неизменяемых данных в произвольном порядке одним или несколькими клиентами; либо в режиме записи данных в файл (и чтения из файла) но только одним клиентом. Несмотря на значительный выигрыш в скорости режим кэширования записи RW-PCC не подойдет для ускорения в среде виртуализации, т.к. в этом случае кэш будет единой точкой отказа (Single-Point-Of-Failure, SPOF): виртуальная ОС не получает гарантии сохранности данных даже после подтверждения записи с помощью операций синхронизации (т.е. вызова fdatasync).

Для параллельных файловых систем были также разработаны системы локального кэширования данных для сглаживания пиковых нагрузок ввода-вывода и уменьшения латентности операций (BurstFS, GekkoFS [21, 22, 23]), но они также не имеют функции отказоустойчивости и поэтому непригодны для сред виртуализации и контейнеризации.

2.1. Критерии оценки систем хранения данных

Многие СХД имеют открытый исходный код и распространяются по лицензиям свободного ПО. Причем исходные коды, как правило, хранятся в публичных репозиториях с системами контроля версий вместе с историей исправлений и внутренней документацией. Это дает возможность изучать устройство этих систем и наблюдать за процессом разработки и сопровождения. Наибольший интерес обычно вызывают: (1) общий размер исходного кода, (2) хроника интенсивности разработки, (3) статистика исправлений разных категорий ошибок, (4) системы внутренних тестов, (5) языки программирования, на которых написаны серверные компоненты, (6) использование внешних инструментов (серверов баз данных, менеджера связности и кворума кластера, менеджера блокировок). Кроме того,

если репозиторий является первичным для разработчиков, то он обычно содержит сообщения об ошибках от сторонних пользователей (issues, pull requests) и ветки с исправлениями и экспериментальными версиями кода (branches, forks).

Для анализа в данной работе было взято несколько распределенных систем хранения с открытым исходным кодом: Coda [18], GlusterFS [20], Lustre [6], OrangeFS [10], Sheepdog [11], XtreemFS [12], Ceph [4], MooseFS [9], EOS [15], ChubaoFS [14], Vitastor [26, 27]. К ним также был добавлен проект Linstor [19], который не является СХД, а служит только «оркестратором» сетевых DRBD-дисков. На рис. 1 изображены результаты анализа протокола изменений в репозиториях выбранных проектов, полученные с помощью команд вида `git log --since … --until … -grep ….` Для каждого проекта определялось общее количество модификаций кода, и количество исправлений, призванных исправить или предотвратить ошибки определенного типа.

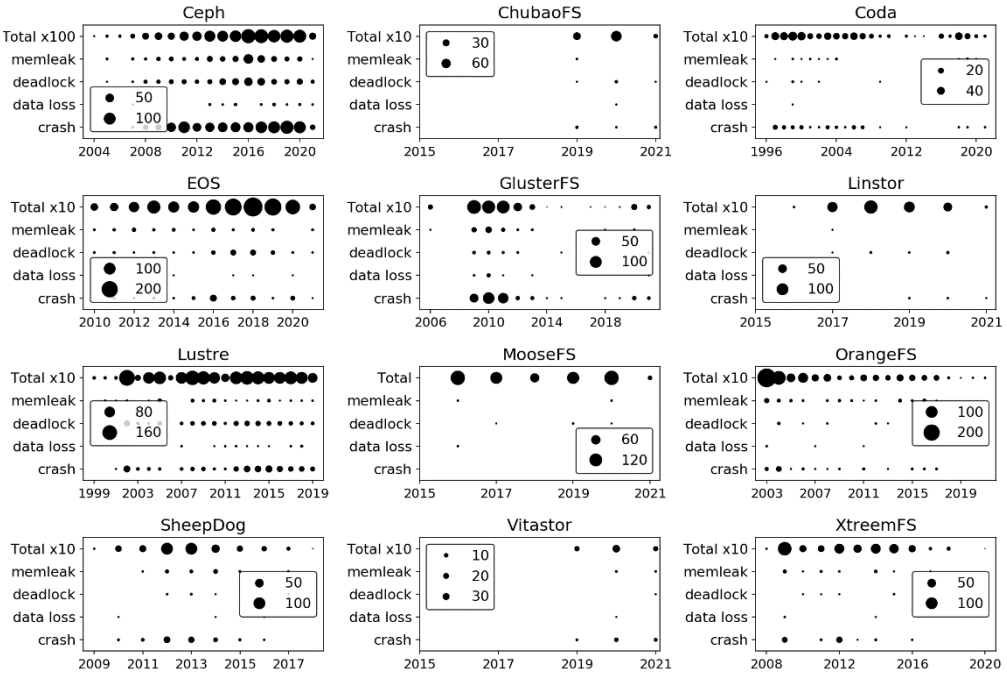

Рис. 1. Динамика разработки двенадцати проектов систем хранения данных с открытым исходным кодом

Типы исправляемых ошибок были классифицированы по характерным словосочетаниям в комментариях: утечка динамической памяти процессов, memleak (словосочетание memory leak в комментарии); ситуации взаимоблокировок разного типа, deadlock (слова deadlock или livelock); потеря хранимых данных, data loss (data loss или data corrupt); аварийное завершение работы; crash (слова crash или panic). Для наглядности показатель общего количества исправлений, Total, уменьшен в 100 раз для Ceph и в 10 раз для остальных проектов, кроме системы MooseFS.

Разумеется, данный анализ подвержен систематическим ошибкам вследствие субъективности процесса комментирования и стиля работы с системой контроля версий. Тем не менее он может предложить базовую гипотезу о качестве кода и стабильности развития проекта. Например, проект Ceph демонстрирует большое общее количество критических ошибок и их увеличение после каждой существенной переделки проекта (таких, как внедрение новой архитектуры OSD-сервера под кодовым названием Bluestore в 2016 году, а затем разработка следующей версии, Crimson, того же сервера в 2017-2020 гг.). Это наблюдение подтверждается сообщениями пользователей [39], утверждающих, что переход на каждую новую версию Ceph с большой вероятностью приводил к сбоям. Параллельная файловая Lustre также показывает высокую интенсивность разработки, однако количество потенциальных проблем существенно меньше. Это подтверждается тем, что Lustre в течение многих лет успешно используется в сегменте высокопроизводительных вычислительных кластеров. Угасание интенсивности разработки обычно означает спад интереса к проекту и уменьшение базы пользователей (это видно на примере систем Sheepdog, GlusterFS, OrangeFS), но может быть вызвано переключением разработчиков

проекта на разработку коммерческой версии ПО с закрытым исходным кодом. Например, проект XtreemFS покинуло несколько разработчиков и основали компанию Quobyte, которая поддерживает коммерческую версию СХД. Система MooseFS продолжает разрабатываться параллельно с коммерческой версией MooseFS Pro.

Серверные компоненты СХД обычно написаны на языках программирования C или C++, хотя XtreemFS полностью написан на Java, а ChubaoFS – на языке Golang. В системе Vitastor разные компоненты написаны на разных языках программирования: сервер монитора, хранящий метаданные и отвечающий на первичные запросы клиентов, является JavaScript-программой в среде интерпретатора Node.js, а сервер хранения данных Vitastor написан на C++.

Суммарный размер программ, т.е. объем исходного кода в строках, и скорость его роста могут также дать дополнительную информацию для оценки текущего состояния проекта и его развития. Например, если остановилось развитие программного проекта небольшого объема, то можно предположить, что он достиг стабильного состояния без нерешенных проблем. Напротив, объемные программные проекты обычно перестают быть работоспособными без поддержки широкого сообщества разработчиков или использующих их коммерческих компаний. Если определить порог для значительного размера в 200000 строк кода, то только Sheepdog, MooseFS, и XtreemFS (см. рис. 2) являются проектами небольшого размера, остановившими свое развитие. Но Sheepdog не получила широкого распространения и по свидетельствам некоторых пользователей имеет нерешенные проблемы стабильности [17], а MooseFS и XtreemFS продолжают развитие в виде коммерческих проектов.

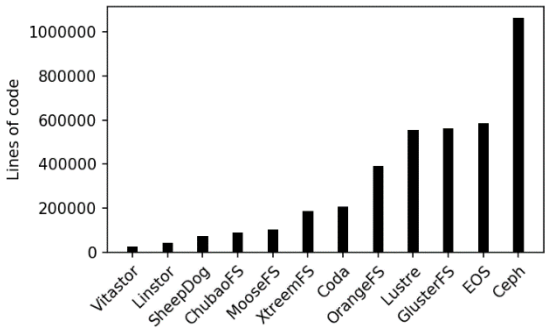

Рис. 2. Объем кода проектов СХД в строках исходных файлов

Среди рассмотренных систем можно также выделить «новое поколение» распределенных СХД: ChubaoFS, EOS, XtreemFS, и Vitastor. Это системы, разработка которых началась совсем недавно, либо которые, хотя и появились более 10 лет назад, но при их разработке были учтены архитектурные недоработки систем предыдущего поколения. Файловая система EOS [15] разрабатывается Европейском центре ядерных исследований CERN в качестве высокопроизводительной системы хранения данных нового поколения для замены системы XRootD [25], также разработанной в CERN и успешно используемой с 2003 года. Разработка XtreemFS была инициирована в Zuse Institute Berlin, ее целью было создание надежной СХД для облачных сервисов. Файловая система ChubaoFS предназначена для крупных систем контейнеризации и разрабатывалась в университете наук и технологий, Хэфэй, КНР. Ее целью было повышение скорости операций с файлами малого и среднего размера по сравнению с существующими распределенными файловыми системами схожего масштаба (CephFS, GlusterFS). Система Vitastor [26, 27] является хранилищем объектов виртуальных блочных устройств, т.е. она является аналогом Ceph RBD, но при этом устроена значительно проще и работает в несколько раз быстрее.

Несмотря на явные различия, эти четыре СХД используют некоторые сходные принципы:

- ослабленные требования к атомарности метаданных, неполная поддержка стандарта POSIX (часто называемые “relaxed POSIX semantics and metadata atomicity”);
- хранение метаданных в отказоустойчивых распределенных key-value базах данных;
- возможность горизонтального масштабирования за пределы кластера локальной сети без значительной потери производительности;
- программный дизайн, ориентированный на работу с быстрыми накопителями, приоритетную оптимизацию операций с данными, и минимизацию потерь производительности в процессе переконфигурирования кластера.

СХД предыдущего поколения также использовали некоторые из этих идей, однако практика показала, что они были реализованы лишь частично. Например, GlusterFS хранит метаданные вместе с данными, а Ceph снижает скорость записи данных практически до нуля во время реконфигурирования кластера после отказа одного из узлов.

В обзоре 2016 года [22] было выполнено сравнение производительности EOS и XtreemFS с некоторыми другими системами хранения данных: GlusterFS, Lustre, BeeGFS, MooseFS,

XRootD, а также с коммерческой системой IBM GPFS (Spectrum Scale). В этих тестах EOS показала сравнимую производительность с Lustre, XrootD, и GPFS в тестах, при работе с одним или несколькими клиентами. В случае одного клиента EOS проигрывает, т.к. она, как и ее прототип, XRootD, не имеют функции striping – автоматического хранения чередующихся сегментов данных на нескольких серверах. XtreemFS, хотя и имеет функцию striping, проигрывает остальным системам во всех тестах. Небольшой прирост скорости от striping в тестах XtreemFS все же наблюдается, но он сопровождается ростом нагрузки на процессоры сервера (которая, тем не менее, не достигает 100%). Можно предположить, что данная неэффективность XtreemFS обусловлена использованием языка Java, в среде интерпретатора которого объекты данных часто претерпевают лишние копирования и тогда производительность ограничивается пропускной способностью оперативной памяти. Необходимо уточнить, что руководство пользователя XtreemFS предупреждает о том, что, хотя эта система и имеет некоторые характеристики параллельной файловой системы, она не претендует на использование в высокопроизводительных вычислительных кластерах. Достоинствами XtreemFS являются возможность простого масштабирования системы за пределы локальной сети и набор встроенных функций для обеспечения отказоустойчивости и стабильной производительности этой системы. Так, для файлов и каталогов можно выбрать одну из нескольких стратегий выбора расположения копий хранимых файлов, одна из которых, fully qualified domain names (FQDN), стремится найти наиболее близкие узлы в дереве сети, тем самым уменьшая задержки при передаче данных и разгружая сетевые коммутаторы.

Отдельно необходимо отметить систему Vitastor [26, 27], которая позиционируется как аналог Ceph RBD, но является значительно более быстрой системой, с простой архитектурой, и оптимизированной для накопителей SSD. Vitastor создана единственным разработчиком и это демонстрирует, что современная распределенная СХД может быть проектом среднего размера. Возможно, причина этого успеха в сочетании простой архитектуры с применением высокоуровневых инструментов (etcd и Node.js) для решения задач управления и мониторинга. Например, разработчик системы Vitastor упростил программы серверов хранения и мониторов, отказавшись от использования в них многопоточности. Система Vitastor имеет драйверы для гипервизора QEMU, расширение для библиотеки Libvirt, а также драйверы для OpenStack (Cinder) и Kubernetes (CSI), поэтому Vitastor может быть протестирована также в качестве хранилища для сред виртуализации и контейнеризации.

Репозитории исходных кодов большинства из рассмотренных СХД содержат наборы тестов. Разумеется, успешное прохождение тестов не доказывает отсутствие ошибок – оно лишь показывает, что не возникают ранее известные ошибочные ситуации. Некоторые тесты разрабатываются специально для проверки выполнения всех ветвей исполняемого кода, но и эта мера не гарантирует корректности в силу огромного размера пространства состояний тестируемой программы. Тем не менее размер и качество тестовой системы является важным индикатором программного проекта.

В некоторых случаях тестирование СХД можно выполнить на одном сервере, т.е. не развертывая кластер. Для этого может быть использована системная виртуализация или изоляция серверных компонентов СХД иными способами, например, методом запуска внутри контейнеров. Из рассмотренных выше СХД такая возможность тестирования предусмотрена только для Sheepdog, Lustre [12], и GlusterFS. Тестирование в таком режиме не позволит оценить рабочие характеристики СХД – такие, как производительность или возможность горизонтального масштабирования, но зато оно даст возможность запуска функциональных тестов, проверяющих реакцию системы на аварийные ситуации. Например, имитация отказа отдельного дискового устройства проверит запуск процесса восстановления набора копий блоков данных. Виртуальные дисковые устройства, созданные средствами драйвера device-mapper ОС Linux, имеют дополнительные функции для имитации разных типов событий: порчи данных (dm-flakey), длительных задержек (dm-delay), ошибок чтения (dm-dust). Возможность имитации таких ситуаций полезна для проверки работы заявленных функций СХД.

2.2. Виды нагрузок и специальные требования клиентов СХД

Многие распределенные файловые системы декларируют свою универсальность, но обычно каждая система имеет свою область применения, на которую она была рассчитана при разработке. С другой стороны, разные типы клиентов файловых систем и СХД могут предъявлять совершенно разные требования к этим системам (см. таблицу 2). Известно, что всем этим требованиям трудно соответствовать одновременно: например, высокую общую пропускную способность трудно совместить с минимальными задержками единичных операций, а частое совместное использование файлов вызывает

нагрузку на механизм блокировок и препятствует масштабированию системы. Поэтому, например, данные СУБД нельзя хранить в файловой системе NAS-сервера.

Набор важных характеристик СХД не ограничивается теми, что приведены в таблице 2. Например, в контексте среды для научных вычислений [24] также рассматривались такие характеристики как: конфиденциальность и важность данных, средний размер файлов, а также избыточность данных, частота их изменения, и частота попадания в локальный кэш. Оказалось, что даже разные задачи одних и тех же пользователей могут сильно различаться по своим требованиям к системе хранения данных: данные результатов вычислительных задач; системные и прикладные исполняемые файлы; «домашние» каталоги пользователей; временные файлы.

Таблица 2. Особенности разных видов использования СХД

Вид использования	Частота совместного доступа	Важность IOPS	Важность общей пропускной способности	Горизонтальное масштабирование
Файловый сервер для пользовательских документов	+	–	–	++
Файлы или блочные устройства для СУБД	–	+++	+++	–
Файлы или блочные устройства для среды виртуализации	–	++	++	+
Архивный сервер (backup server)	–	–	++	+
СХД для вычислительного кластера*	–	++	+++	++
Файловый сервер для web-хостинга**	+	++ (чтение)	++ (чтение)	+++

(*) Интенсивность ввода-вывода и обмена данными между узлами вычислительного кластера зависит от типа решаемой задачи. Примерами задач с разными потребностями файлового ввода-вывода могут служить алгоритм быстрого преобразования Фурье, умножение матриц и оптимизационная задача класса NP.
(**) Системы хранения и доставки контента для крупных web-серверов имеют дополнительные требования, см. обсуждение ниже.

2.3. Модели консистентности данных

Другими двумя важными характеристиками распределенных СХД являются модель консистентности данных и набор атомарных операций записи. Моделями консистентности в контексте СХД называют наборы условий, которые выполняются для всех сценариев работы независимых пользователей СХД. На практике это означает, в каком порядке пользователь может увидеть изменения, вносимые в данные им и другими пользователями СХД. В контексте распределенных СХД принято различать несколько моделей консистентности данных [25], хотя обычно выбор сводится к трем альтернативам:

- модель *строгой консистентности* данных, согласно которой результат всех операций ввода-вывода соответствует одному общему плану, который не нарушает порядка операций, выполняемых каждым отдельным клиентом;
- модель *отложенной консистентности* данных (eventual consistency model), в которой допускается, что порядок выполнения операций с точки зрения разных клиентов отличается, но через некоторое ограниченное время после последней модификации данных все клиенты будут видеть одно и то же состояние всех объектов;
- модель *слабой консистентности* данных, которая не дает никаких конкретных гарантий, но может иметь для поиска и исправления расхождений дополнительные инструменты копий данных на узлах СХД.

В модели строгой консистентности данных любая потеря связности множества узлов системы приводит к приостановке обслуживания клиентов, поскольку иначе нельзя гарантировать корректность результата всех операций. Модель отложенной консистентности данных формально не гарантирует строгой очередности операций с объектом даже при монопольном доступе к объекту данных, хотя в большинстве таких систем гарантируется, что клиент при чтении получает результат последней записи, а операции записи выполняются в том же порядке, в котором они были переданы системе.

Кроме модели консистентности необходимо также определять гранулярность атомарных операций модификации данных, которые могут пересекаться и образовывать конфликт. Операции могут затрагивать весь файл (или объект), отдельные страницы данных, или произвольный диапазон байтов. Модификации непересекающихся (с учетом гранулярности) областей данных в файле не образуют конфликта, поскольку

в набор разрешенных операций не входят операции вставки и удаления данных в произвольной позиции – по историческим причинам содержимое файлов может быть лишь изменено в произвольной позиции, а новые данные могут быть добавлены только в конец файла.

В контексте файл-серверов также применяется модель слабой консистентности, называемая close-to-open, согласно которой содержимое файла, обновленное одним клиентом, видно другим клиентам только после того, как работа с этим файлом была завершена и файл был открыт для чтения. Модель close-to-open стала популярна, поскольку используется сетевой файловой системой NFS, а среди рассмотренных выше СХД эту модель применяют системы XtreemFS и OrangeFS.

Наиболее известным примером реализации строгой модели консистентности является набор и семантика операций с файлами в стандарте POSIX. Эта модель достаточно строгая, чтобы ограничивать рост производительности даже в кластерах небольшого размера. Кроме того, в стандарте POSIX отсутствуют требования обеспечения атомарности операций записи данных, которые нужны как прикладным программам, так и конечным пользователям. Например, не определен результат чтения документа электронной таблицы в момент, когда этот файл перезаписывается другим пользователем, однако файл-сервер не обеспечивает атомарности действий на уровне файлов. Чтобы избегать таких ситуаций, прикладным программам приходится использовать дополнительные средства синхронизации доступа – например, lock-файлы для блокировки одновременного доступа или запись во временный файл с последующим атомарным переименованием. Трудности, возникающие при работе с хранением данных в POSIX-совместимых ОС широко известны [41].

В распределенной СХД соблюдение модели консистентности данных обеспечивается обычно действиями самой системы, но иногда предполагается, что клиенты также выполняют определенные действия, необходимые для исключения конфликтов доступа или для упрощения и ускорения работы ввода-вывода. Например, Lustre для достижения строгой консистентности данных использует механизм распределенных блокировок, и поэтому явные блокировки файлов для записи (вызовы функции flock) не являются обязательными. Однако клиенты все-таки могут устанавливать блокировки специальных типов для ускорения доступа параллельных программ к одному файлу, например, когда паттерн доступа строго определен и клиент берет на себя ответственность за исключение конфликтов. При использовании Lustre вне контекста параллельных вычислений также рекомендуется соблюдение некоторых правил, чтобы уменьшить влияние механизма блокировок на производительность. Например, следует выключать размещение с чередованием для каталогов, в которых хранятся файлы небольшого размера (настройка stripe count = 1), и не создавать каталоги с большим количеством файлов. Некоторые системы, наоборот, по умолчанию не соблюдают строгую консистентность данных, но предлагают механизм разграничения доступа. Например, система MooseFS требует, чтобы в случае совместного доступа к файлам клиенты использовали механизм блокировок стандарта POSIX. Наконец, в случаях использования модели слабой консистентности механизм распределенных блокировок может либо полностью отсутствовать, либо допускать ситуации, в которых после потери связности кластера его компоненты продолжают модифицировать общие объекты. В таком случае проблема конфликта независимых модификаций данных откладывается до момента его обнаружения, а затем передается на уровень прикладных программ. Примером использования такой модели служит система Coda [27].

Иногда модель строгой консистентности следует из самого алгоритма репликации данных. Например, на базовом уровне системы Ceph RADOS [5] все модификации объектов атомарные в силу механизма работы серверов хранения данных, а в случае аварии чтение устаревших копий объектов исключается особым режимом работы, который используется в процессе восстановления пула. Поэтому Ceph называют системой, гарантирующей строгую консистентность данных, хотя на уровнях сервера блочных устройств Ceph RBD и файловой системы CephFS должны использоваться блокировки, иначе результат совместного доступа к одному и тому же устройству или файлу будет непредсказуем.

Для построения производительных и масштабируемых прикладных систем важно определять лишь минимально необходимые требования к системе хранения данных. В том случае, когда СХД используется для хранения объектов относительно небольших размеров, то согласованность сложных данных часто можно обеспечить, пользуясь только гарантиями атомарности операций над элементарными объектами и слабой моделью консистентности. Например, если интернет-страница сервиса web-хостинга состоит из HTML-файла и сопутствующих ему ресурсов (CSS-стиль, изображения и пр.), то, чтобы обновить страницу, нужно сначала создать новые версии этих ресурсов, а затем новую

версию HTML-файла с обновленными ссылками. Если система хранения гарантирует очередность создания объектов (например, правило ‘strong read-after-write consistency’ в терминах сервиса Amazon AWS S3), то интернет-страница во все моменты времени будет представлена в консистентном состоянии. Если же очередность появления созданных объектов не гарантируется, то можно либо задержать обновление HTML-файла на время, которое в среднем требуется СХД для распространения изменений в дата-центре, либо не делать даже этого, тогда в худшем случае пользователь увидит неполную интернет-страницу, которую он должен будет перезагрузить вручную.

Приведенные выше примеры показывают, что построение универсальной распределенной отказоустойчивой системы хранения данных со строгой моделью консистентности не только практически невозможно, но и не требуется в большинстве случаев, когда нужно создать масштабируемый сервис для хранения и несложной обработки данных.

2.4. Проблемы неточного определения модели хранения данных

Распределенные системы хранения данных, являясь сервисами со сложной архитектурой, обычно не предоставляют пользователям подробной информации о своем устройстве и состоянии. В частности, пользователям редко известны все важные свойства используемой ими СХД, а именно:

Протокол подтверждения записи данных. Момент записи данных в накопитель не всегда точно определен и может зависеть от программных и аппаратных настроек компонентов СХД. Кроме того, момент репликации данных также может быть отложен (например, при использовании протоколов записи A и B в системе Linstor/DRBD);

Алгоритм восстановления данных при выходе из строя накопителя или узла хранения. Восстановление массива данных может запускаться немедленно или откладываться до момента простоя, есть ли ограничение максимальной скорости репликации. Гранулярность механизма репликации может быть также разная (блок данных, файл, том). От этих факторов зависят риски потери данных и производительность СХД в нештатных ситуациях;

Алгоритм работы в случае потери связности кластера. Временный или постоянный обрыв связи между узлами вызывает отложенную или немедленную переконфигурацию распределенной СХД и следующую за ней репликацию данных. Выбор той или иной стратегии имеет как преимущества, так и недостатки. Дополнительные проблемы может вызвать восстановление связи с изолированным узлом, если система разрешает обратное подключение отказавшего узла (это одна из известных проблем системы Ceph);

Контроль целостности данных и восстановление ошибок. Скрытые ошибки данных (silent data corruption, bitrot), редко возникающие при работе накопителей информации, могут либо оставаться незамеченными, либо выявляться и оперативно исправляться. Несмотря на экспоненциальный рост объема накопителей данных, приводящий к увеличению вероятности таких ошибок, только некоторые из современных систем могут выполнять контроль и исправление таких ошибок – например, локальная файловая система ZFS и СХД Ceph (начиная с первой версии сервера хранения данных Bluestore).

На практике только системные администраторы, эксплуатирующие СХД, обладают достаточным знанием о текущем состоянии системы и алгоритме ее работы в нештатных ситуациях. Прикладные программы и пользователи вынуждены полагаться на существующую документацию и опыт, полученный в процессе эксплуатации системы. Кроме того, такое недостаточное определение интерфейса и алгоритмов работы СХД могут быть причиной логических ошибок, которые ведут к потерям данных или критическому снижению производительности.

С другой стороны, системы хранения данных тоже страдают от недостатка информации, т.к. они могли бы использовать подсказки от клиентов о назначении хранимых данных и предполагаемом виде нагрузки ввода-вывода для оптимизации алгоритмов работы и улучшения качества обслуживания. Частично эту информацию можно сообщить, используя стандартные функции POSIX-стандарта (fcntl, posix_fadvise, ioctl, mmap + madvise), но многие СХД для администрирования системы и пользовательских настроек предлагают также использовать специальные утилиты командной строки, которые доступны не только администраторам, но и обычным пользователям.

При использовании специализированных библиотек для работы с СХД появляется возможность выполнять операции, не предусмотренные стандартом POSIX, например, с помощью библиотеки Ceph librados можно выполнять транзакции, состоящие из нескольких элементарных операций модификации данных в объектном хранилище RADOS. Для некоторых СХД интерфейс специальной библиотеки является основным для работы прикладных программ с данными. В этом случае не нужно представлять хранимые данные в виде файловой системы и поэтому не требуется взаимодействие с ядром

ОС. Это, в свою очередь, упрощает адаптацию клиентской библиотеки к среде новых ОС. Наиболее известным примером такой системы служит Hadoop distributed file system, HDFS, которая выполняет функцию хранилища данных в среде системы кластерных вычислений Apache Hadoop.

Учитывая вышеизложенное, можно предположить, что отсутствие общепринятого стандарта описания семантической модели работы распределенных СХД негативно сказывается на процессах их разработки и эксплуатации.

2.5. Скоростные системы хранения на базе накопителей стандарта NVMe SSD

Появление накопителей нового поколения NVMe SSD придало новый стимул для развития распределенных СХД. Такие накопители обеспечивают существенно более высокую пропускную способность и малые задержки для доступа к данным, поэтому при использовании в качестве локального накопителя на узле кластера они оказываются быстрее сети хранения данных (SAN), и тем более быстрее распределенных СХД. Но необходимость дупликации или избыточного кодирования данных остается, поскольку NVMe-накопители тоже подвержены отказам и имеют ограниченный ресурс записи. Это заставляет продолжать применять централизованные массивы устройств либо распределенные системы хранения.

Однако при попытке использования новых устройств в традиционных СХД часто оказывается, что производительность системы сдерживается уже не скоростью устройств хранения данных, а вспомогательными элементами: сетями передачи данных и процессорами узлов хранения. Например, программный код систем, разработанных для устройств HDD и SSD первых поколений, мог не учитывать цену переключений контекста процессора из пользовательского режима в режим ядра и обратно. Автор Vitastor выполнил тесты и показал [38], что производительность системы хранения Ceph на современном оборудовании сдерживается исключительно неоптимальной программной архитектурой серверов, хранящих блоки данных (Bluestore), и это не может быть исправлено никакими настройками системы. Такая ситуация относительно скоростей устройств и текущего состояния ПО СХД снова дает шанс гиперконвергентным системам для захвата рынка, если они смогут доказать свою способность эффективно обеспечивать локализацию данных по отношению к потребителям.

Разрыв в производительности между единичным накопителем и системами хранения данных стимулирует и производителей аппаратуры разрабатывать технологии, помогающие строить большие СХД без многократного проигрыша в скорости передачи данных. Так, новая версия спецификации NVMe 2.0 [39] содержит:

- Переработанное описание транспортного уровня, в котором протоколы PCIe, RDMA и TCP представлены равноправно;
- Пространства имен с функциями зонирования (Zoned Namespaces), которые заменяют встроенный модуль трансляции страниц памяти (Flash Translation Layer, FTL) более гибким механизмом управления записью данных в устройстве;
- Стандарт для описания вычислительных функций накопителя данных (NVMe Computational Storage);
- Интерфейс встроенных «key-value» хранилищ данных.

Кроме того, уже предыдущие версии спецификации NVMe позволяли использовать Multipath I/O для подключения устройств к нескольким контроллерам для достижения отказоустойчивости и балансировки нагрузки.

Скорости современных сетевых карт и NVMe-устройств приближаются к скоростям обмена данных по шинам PCIe и скоростям доступа к оперативной памяти в NUMA-архитектуре. Поэтому при проектировании устройства серверов хранения данных нужно анализировать соответствие расположения NVMe-накопителей и сетевых карт топологии шин PCIe и оценивать запас пропускной способности всех каналов передачи данных. Более того, высокая частота событий, связанных с обработкой данных, делает неэффективным использование обычного механизма оповещения драйвера устройства с помощью аппаратных прерываний и переключений в режим ядра ОС. Чтобы сократить количество переключений контекста процессора, предлагается либо вынести всю работу с устройством из контекста ядра, например, с помощью библиотеки Intel SPDK [30], либо использовать системный вызов io_uring – новый механизм ядра Linux для асинхронной и пакетной обработки событий ввода-вывода, который существенно сокращает накладные расходы по координации действий драйвера устройства и прикладной программы. Примером оптимального конфигурирования и настройки сервера может служить описание эксперимента [40], где рабочая станция с 16-ядерным процессором AMD Ryzen Threadripper PRO 3955WX и восемью NVMe-накопителями Samsung 980 Pro PCIe 4.0 M.2 показала на тесте чтения данных 11 миллионов IOPS и общую

пропускную способность 66Гб/сек. Однако данная конфигурация не содержала сетевых карт, которые необходимы для доставки этого потока данных клиентам. Полезные рекомендации по подбору и настройке высокоскоростных серверов хранения данных можно найти у разработчиков коммерческих программных SAN-массивов (StarWind [32], StorPool [33]), гиперконвергентных (Nutanix [34]), и распределенных систем хранения данных (WekaFS [25]).

В настоящее время только у производителей SAN- и NAS-систем СХД (DELL EMC All-Flash, HPE Nimble, IBM FlashSystem, NetApp SolidFire all-flash, PureStorage FlashArray, и др.) и провайдеров услуг облачных вычислений (Amazon AWS, Google Compute Platform, Microsoft Azure, Alibaba Cloud, и др.) есть значительный опыт консолидации NVMe-накопителей и построения на их основе масштабируемых скоростных хранилищ данных. Но поскольку этот опыт дает конкурентное преимущество, то за редкими исключениями [23, 45] детали реализации этих СХД полностью закрыты. Тем не менее, если технология NVMe-oF получит такую же популярность, какую в свое время получили технологии iSCSI и Fiber Channel, то можно ожидать появления нового поколения централизованных и распределенных систем хранения данных, в том числе и проектов с открытым исходным кодом.

3. Алгоритм выбора и внедрения распределенной СХД

Решение об использовании той или иной СХД не всегда легко обосновать, поскольку часто вопреки общим тезисам о масштабируемости и универсальности, такие системы имеют разное устройство и разные области применимости. Иногда задача выбора СХД искусственно упрощается с помощью апелляции к недоказанным утверждениям и переключением внимания на маловажные критерии. Например, авторы считают следующие аргументы в пользу использования распределенных СХД *неверными*:

Универсальное хранилище данных как замена локальных ФС. Распределенные ФС не обеспечивают модели консистентности данных и производительности операций, соответствующие локальным файловым системам. Кластерные ФС (например, GFS2 и OCFS2) наиболее близки по свойствам к локальным ФС, но они практически не масштабируются и их производительность сильно деградирует при определенных типах нагрузок.

Использование распределенной СХД вместо архива. Распределенные СХД не служат заменой файловым архивам. Такие системы можно использовать для хранения архивных данных, но для управления архивами нужно использовать специальное ПО.

Обеспечение отказоустойчивости. Масштабируемые распределенные СХД, как правило, устойчивы к отказам узлов, но эта функция не должна быть основным стимулом к их использованию. Если потребности пользователей и приложений в вычислительном кластере могут быть удовлетворены сочетанием SAN- и NAS-серверов, то обеспечить их высокую устойчивость к отказам оборудования можно, дублируя набор сетевых коммутаторов, цепей энергопитания, и объединяя накопители в RAID-массивы, дополнительно повышая надежность их подключения с помощью Multipath I/O.

Более обоснованы следующие аргументы в пользу использования распределенных СХД:

Горизонтальное масштабирование. Возможность наращивания числа узлов хранения без значительного снижения скорости доступа и с сохранением отказоустойчивости.

Защита данных от повреждения. С увеличением объема хранимых данных повышается вероятность отказов устройств и появление скрытых ошибок данных. Распределенные СХД, как правило, используют более гибкие алгоритмы контроля целостности данных и их восстановления из частично поврежденных копий.

Локальность хранения данных. Некоторые распределенные СХД имеют функцию управления расположением копий данных. Эта функция может ускорять доступ к данным для потребителей в крупных сетях, например, в системе доставки статического контента крупных web-порталов. Для инфраструктуры облачных вычислений локальность данных увеличивает общую производительность и ускоряет миграцию виртуальных ОС.

Дополнительные функции. Многие распределенные СХД предоставляют возможности, недоступные пользователям в традиционных ФС: моментальные снимки каталогов и файлов, атомарные транзакции над объектами данных, обязательные блокировки файлов. Параллельные ФС и СХД с децентрализованной архитектурой обеспечивают высокую скорость доступа за счет прямого доступа клиентов к узлам хранения данных.

Задача выбора и внедрения распределенной СХД обычно состоит из этапов оценки требований, предъявляемых к системе, выбора доступных альтернатив, предварительного тестирования, и пробной эксплуатации выбранной системы.

3.1. Оценка требований клиентских приложений

Требования к производительности не всегда

можно точно определить, но если производительность системы недостаточна для работы приложений, то это обнаруживается при тестировании. Требования надежности и отказоустойчивости сложнее сформулировать и гораздо сложнее проверить, учитывая количество сочетаний внешних факторов и последовательности событий в распределенной вычислительной системе. Проекты с открытым исходным кодом и большой базой пользователей обычно имеют накопленные сведения об известных нештатных ситуациях и ошибках. Знание этих потенциальных проблем помогает составить план тестирования отказоустойчивости и избегать некорректных конфигураций системы.

Если систему планируется использовать для нескольких приложений, оказывающих разную нагрузку, то для тестирования следует выбрать основное приложение, предъявляющее наивысшие требования к системе. Когда такое приложение выделить нельзя, тесты будут более сложными, и, соответственно, их результаты будет труднее интерпретировать. Кроме того, сочетание разных видов нагрузок часто приводит к снижению производительности СХД (возникает перегрузка сервера метаданных, задержки в системе взаимных блокировок, и пр.). Документация систем СХД может содержать рекомендации для диагностики и устранения таких ситуаций.

Если допустима адаптация приложений, то она может помочь унифицировать доступ к системе и упростить ее настройку. Например, переход от использования файловой системы к объектному хранилищу увеличивает производительность и упрощает масштабирование, но при этом предполагает отказ от модели строгой консистентности данных. В случаях, когда комбинированная нагрузка все-таки неизбежна, решением может стать применение нескольких подсистем, каждая из которых имеет архитектуру и настройки, соответствующие требованиям ее клиентов. Такая конфигурация позволит независимо оценить объемы данных, хранимых в каждой из подсистем, и спланировать их будущее масштабирование. В некоторых случаях после такого разделения оказывается, что применять распределенные системы хранения уже не обязательно, т.к. каждая из подсистем помещается в одном отказоустойчивом кластере, подключенном к SAN-хранилищу.

На стадии выбора распределенной СХД часто труднее всего определить потребность в будущем масштабировании системы. Причина в том, что скорость роста объема данных СХД часто зависит от интенсивности использования этой системы на начальной стадии эксплуатации, т.е. от успешности ее внедрения. Иногда это приводит к парадоксальным ситуациям, когда скоростное хранилище данных, установленное для хранения «горячих» данных, используется как универсальное и переполняется, тогда как распределенная система хранения данных показывает низкую производительность, и поэтому остается незагруженной.

Особого рассмотрения требуют клиенты, которые ожидают от системы хранения данных производительности, сравнимой с быстрыми локальными накопителями. Примерами таких клиентов служат однопотоковые приложения с синхронными операциями ввода-вывода и виртуальные системы с устаревшими версиями ОС. Распределенные системы хранения могут обеспечить высокие значения общей пропускной способности (bandwidth) и количества операций (IOPS), но задержка единичных операций оказывается в 10-100 раз выше. Обычно для ускорения работы таких приложений либо применяют кэширование, либо настраивают прямой доступ к накопительным устройствам (например, см. анализ производительности ввода-вывода в среде QEMU/KVM [2]). Кэширование в операциях чтения и опережающее чтение эффективны, если приложения используют данные монопольно, или интенсивность обмена данными через общие файлы невелика. Кэширование операций записи, напротив, сопряжено с рисками потери данных и проблемами синхронизации кэшей в распределенных системах. Так, включение режима отложенной записи (writeback) для кэша виртуальных дисков в среде виртуализации допустимо только после подтверждения его безопасности со стороны клиентов, т.к. многие устаревшие приложения неявно предполагают, что завершенная операция записи гарантирует помещение данных на носитель. Если же прикладная система избыточно часто делает вызовы функции синхронизации данных, то кэширование записи будет безопасным, но не даст увеличения производительности.

Другим способом повышения скорости операций записи является организация прямого доступа к локальному накопителю с отложенной синхронизацией реплик данных на другие узлы. Такой подход используют гиперконвергентные системы и система управления дисковыми томами Linstor (если для синхронизации данных применяется протокол A или B, см. [3, 19]). Однако следует понимать, что в таких системах гарантии сохранности данных снижены, т.к. приоритет отдан достижению максимальной скорости. Наконец, с увеличением нагрузки эффективность кэширования снижается, а отложенная синхронизация перестает справляться с потоком данных и начинает задерживать очередь операций записи. Поэтому, рассматривая подобные решения, необходимо предусмотреть будущий

рост числа виртуальных ОС или экземпляров приложений на узлах кластера, и заранее протестировать работу системы в режиме максимальной нагрузки.

Резюмируя, можно сказать, что если среди потенциальных клиентов СХД есть приложения, требующие высоких скоростей ввода-вывода, то на данный момент лучше подключить эти приложения к AFA флэш-массиву, как к централизованному NAS- или SAN-хранилищу. Позже, в случае успешной работы этих приложений, придется либо нарастить объем этого массива, либо повторно рассмотреть возможность использования распределенных СХД.

3.2. Тестовый стенд и пробная эксплуатация СХД

Тестирование выполняет две задачи: проверку принципиальной возможности использования СХД и оценку ее производительности. Стенд должен соответствовать минимальным требованиям системы, но рекомендуется, чтобы он состоял из того же оборудования, которое планируется использовать позже. Кроме того, на стенде минимального размера трудно проверить отказоустойчивость системы: для этого нужны резервные узлы хранения данных и каналы обмена данными, а также, возможно, отдельная сеть для управления кластером. Прежде, чем начать установку и тестирование СХД, рекомендуется выполнить стресс-тесты всех вычислительных модулей, накопителей данных, сетевых интерфейсов, и коммутаторов. Эти тесты не только проверят надежность оборудования, но и определят пиковые показатели скоростей записи, чтения, и передачи данных. Рекомендации по выбору оборудования можно найти в документации тестируемой СХД, публичных отчетах о проведенных тестированиях [13, 24], а также в рекомендациях производителей программных коммерческих систем [22, 23]. Кроме того, рекордные показатели производительности ввода-вывода в среде вычислительных кластеров можно найти в отчетах Supercomputing 2020 [26] (где также приведены результаты конкурса для небольших кластеров, см. секцию 10 Node Challenge) и SPEC Storage 2020 [37].

Если СХД допускает установку в среде виртуализации, то некоторое тестирование и общее ознакомление с системой можно выполнить без развертывания стенда. Тем не менее, такие эксперименты не могут дать утвердительного ответа о применимости системы, поскольку они не позволяют оценить ни скорость, ни надежность при работе с реальным оборудованием. Также не рекомендуется, не завершив тестирования, начинать использовать стенд для полезной нагрузки, т.к. тогда во избежание потери данных тестирование отказоустойчивости будет свернуто. Следует отметить, что программы с открытым исходным кодом и свободной лицензией дают больше возможностей для их изучения и тестирования, чем аналогичные им коммерческие продукты, тестирование которых перед покупкой либо невозможно, либо сильно ограничено.

4. Заключение

Распределенные СХД эволюционируют вместе со сменой технологий, увеличением общего объема хранимых данных, и изменением запросов пользователей. Современное оборудование способно обеспечить существенно более высокие скорости доступа к данным, но масштабирование таких систем хранения с сохранением производительности и надежности сталкивается с ограничениями использованных программных архитектур. Развитие распределенных хранилищ данных с открытым исходным кодом запаздывает по отношению к коммерческим программно-аппаратным СХД и сервисам хранения данных провайдеров облачной инфраструктуры.

Тем не менее разработке открытых систем должна помочь стандартизация оборудования и протоколов обмена данных. Особенно полезна была бы стандартизация описания семантических моделей этих систем, поскольку неполное понимание алгоритмов их работы приводит к логическим ошибкам в системном и прикладном ПО. В стандартизации должны быть заинтересованы некоторые провайдеры облачной инфраструктуры и разработчики коммерческих программных СХД. Производители аппаратных СХД и крупные провайдеры – напротив, могут использовать собственные нестандартные решения для оптимизации протоколов с целью получения конкурентных преимуществ, одновременно привязывая пользователей к своей платформе.

Если рассматривать доступные на данный момент СХД с открытым исходным кодом, то оказывается, что системы, доказавшие свою стабильность (Lustre, Ceph, GlusterFS) не полностью соответствуют современным требованиям к скорости и надежности, тогда как более современные системы еще не имеют пользовательской базы, достаточной для уверенности в их будущем развитии и сопровождении. В этих условиях рекомендуется использовать поэтапную стратегию выбора СХД, оценивая требования каждого клиента и используя распределенные системы хранения после оценки преимуществ и недостатков их использования по сравнению с традиционными централизованными хранилищами.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН (Проведение фундаментальных научных исследований (47 ГП) по теме № FNEF-2021-0007 «Развитие методов математического моделирования распределенных систем и соответствующих методов вычисления. 0580-2021-0007», Рег.№ 121031300051-3).

Modern Distributed Storage Systems: Feasibility Criteria and Testing Methodology

A.B. Betelin, I.B. Egorychev, A.A. Prilipko, G.A. Prilipko, S.G. Romanyuk, D.V. Samborskiy

Abstract. This article provides an overview of modern open source distributed storage systems. Scalable storage systems gain popularity due to the growing needs for storing and analyzing large amounts of data. The problem of choosing a specific system is complicated by the variety of software architectures used in these systems and their functional features. The article analyzes the main factors that determine the compliance of the data storage system with given requirements, and proposes a methodology for its testing and deployment.

Keywords: distributed storage system, file system, horizontal scaling

Литература

1. E.A. Brewer. Towards robust distributed systems. «InPODC 2000», Vol. 7, No. 10.
2. А.Б. Бетелин, И.Б. Егорычев, А.А. Прилипко, Г.А. Прилипко, С.Г. Романюк, Д.В. Самборский. Настройка и оптимизация системы ввода-вывода в среде виртуализации GNU Linux/QEMU/KVM/Libvirt. «Труды НИИСИ РАН», Том 9 (2019), № 5, 119-129.
3. А.Б. Бетелин, А.А. Прилипко, Г.А. Прилипко, С.Г. Романюк, Д.В. Самборский. DRBD как средство для создания устройств хранения данных в среде ОС GNU Linux. «Труды НИИСИ РАН», Том 9 (2019), № 2, 50-58.
4. S.A. Weil. Ceph: reliable, scalable, and high-performance distributed storage. University of California, Santa Cruz, 2007.
5. S.A. Weil, et al. Rados: a scalable, reliable storage service for petabyte-scale storage clusters. Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07.
6. Peter, J. The lustre storage architecture. Cluster File Systems, Inc (2004).
7. Y. Qian, X. Li, S. Ihara, A. Dilger, C. Thomaz, S. Wang, A. Brinkmann. LPCC: Hierarchical persistent client caching for lustre. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2019).
8. Сайт “KVM Quick Start Guide - Lustre Wiki”, https://wiki.lustre.org/KVM_Quick_Start_Guide
9. Сайт проекта MooseFS, https://moosefs.com
10. M.M. Bonnie, et al. OrangeFS: Advancing PVFS. «USENIX Conference on File and Storage Technologies (FAST) 2011»
11. Сайт проекта Sheepdog, https://sheepdog.github.io/sheepdog
12. Сайт проекта XtreemFS, http://www.xtreemfs.org
13. D. Autiero, D. Caiulo, S. Galymov, J. Marteau, E. Pennacchio, E. Bechetoille, B. Carlus, C. Girerd, H. Mathez. Comparison between different online storage systems, WA105 Technical Board Meeting, June 15th, 2016
14. H. Liu, et al. Cfs: A distributed file system for large scale container platforms. Proceedings of the 2019 International Conference on Management of Data. 2019.
15. Adde G., Chan B., Duellmann D., Espinal X., Fiorot A., Iven J., Janyst L., Lamanna M., Mascetti L., Rocha J.M.P., Peters A.J. Latest evolution of EOS filesystem. In Journal of Physics: Conference Series 2015 (Vol. 608, No. 1, p. 012009). IOP Publishing.
16. A. Dorigo, P. Elmer, F. Furano, A. Hanushevsky. XROOTD-A Highly scalable architecture for data access. WSEAS Transactions on Computers, 2005 1(4.3), pp.348-353.
17. А. Квапил. В поисках идеального хранилища... «Saint HighLoad++ 2021».

18. P.J. Braam, et al. Porting the Coda File System to Windows. «USENIX Annual Technical Conference», FREENIX Track. 1999.

19. Сайт "Linstor Volume Manager", https://linbit.com/software-defined-storage

20. Сайт проекта GlusterFS, https://www.gluster.org

21. T. Wang, K. Mohror, A. Moody, K. Sato, W. Yu. An ephemeral burst-buffer file system for scientific applications. In Proc. the 2016 International Conference for High Performance Computing, November 2016, pp.807-818.

22. M. Vef, N. Moti, T. Süß, T. Tocci, R. Nou, A. Miranda, T. Cortes, A. Brinkmann. GekkoFS — A temporary distributed file system for HPC applications. In Proc. the 2018 IEEE Int. Conf. Cluster Computing, September 2018, pp.319-324.

23. A. Brinkmann, et al. Ad hoc file systems for high-performance computing. Journal of Computer Science and Technology 35.1 (2020): 4-26.

24. J. Blomer. Experiences on File Systems: Which is the best file system for you? CERN PH/SFT, CHEP 2015, Okinawa, Japan.

25. D. Bermbach, J. Kuhlenkamp. Consistency in distributed storage systems. «International Conference on Networked Systems» (pp. 175-189). Springer, Berlin, Heidelberg.

26. Сайт “Vitastor – надёжное кластерное блочное хранилище без единой точки отказа”, https://vitastor.io

27. В. Филиппов. Vitastor, или как я написал свою хранилку. «DevOps Conf 2021», https://vitastor.io/presentation/devopsconf/devopsconf.html

28. Сайт “Производительность Ceph”, https://yourcmc.ru/wiki/Производительность_Ceph

29. Сайт разработчиков спецификации NMV Express, https://nvmexpress.org/developers/nvme-specification

30. Сайт проекта SPDK, https://spdk.io/doc

31. Сайт “Hyper-Converged Infrastructure Production Record: 20 million IOPS. 84% of the theoretical limit”, https://www.starwindsoftware.com/nvme-of-working-in-a-12-node-all-nvme-cluster

32. Сайт “StorPool System Requirements”, https://goo.gl/oBsESr

33. Сайт “Nutanix HCI: Supported Hardware Platforms and Public Clouds”, https://www.nutanix.com/uk/products/hardware-platforms

34. X. Shuai, et al. Spool: Reliable Virtualized NVMe Storage Pool in Public Cloud Infrastructure. «USENIX Annual Technical Conference» (2020)

35. Сайт “WEKA: Prerequisites and Compatibility”, https://docs.weka.io/support/prerequisites-and-compatability

36. Сайт “Supercomputing 2020 benchmark results”, https://www.vi4io.org/io500/list/20-11/start

37. Сайт “SPECstorage Solution 2020 Results”, https://www.spec.org/storage2020/results

38. Информационная технология - интерфейс мобильной операционной системы (POSIX) - Часть 1: Интерфейс прикладных программ (API) [Язык программирования C], «Международный стандарт ISO/IEC 9945-1: 1996(E)», Издание НИИСИ РАН Москва 1999

39. Сайт “Как мы отказоустойчивый кластер запускали”, https://habr.com/ru/company/-first/blog/314106

40. Сайт “Achieving 11M IOPS & 66 GB/s IO on a Single ThreadRipper Workstation” (https://tanelpoder.com/posts/11m-iops-with-10-ssds-on-amd-threadripper-pro-workstation)

41. Сайт “What’s So Bad About POSIX I/O?” https://www.nextplatform.com/2017/09/11/whats-bad-posix-io

HIGGS – утилита резервного копирования данных с поддержкой сжатия и дедупликации

А.Б. Бетелин[1], И.Б. Егорычев[2], А.А. Прилипко[3], Г.А. Прилипко[4], С.Г. Романюк[5], Д.В. Самборский[6]

[1]ab@niisi.msk.ru, [2]egorychev@niisi.msk.ru, [3]aaprilipko@niisi.msk.ru, [4]prilipko@niisi.msk.ru, [5]sgrom@niisi.ras.ru, [6]samborsky_d@fastmail.com;

ФГУ ФНЦ НИИСИ РАН, Москва, Россия

Аннотация. Резервное копирование больших объемов неструктурированных данных в средах виртуализации и системах управления базами данных необходимо для возможности восстановления работы прикладных систем после неустранимого сбоя или случайной порчи данных. Низкий процент измененных данных приводит к высокой степени избыточности копий, если не использованы методы оптимизации, такие как создание цепочек инкрементальных архивов или поиск общих фрагментов данных (дедупликация). В статье представлена утилита универсального архиватора Hard link-Inspired Granular Gracious Storage (HIGGS), специально разработанная авторами данной работы для эффективного хранения резервных копий произвольных бинарных данных. Тестирование производительности программы HIGGS показало ее преимущество по сравнению с несколькими популярными утилитами резервного копирования данных, также имеющими функцию дедупликации.

Ключевые слова: архивирование, дедупликация, параллельные программы, жесткие ссылки, сжатие данных

1. Введение

Регулярное резервное копирование данных позволяет восстановить работоспособность прикладных систем в случае отказа оборудования или случайной потери данных. Формально термины «резервное копирование» и «архивирование» означают разные действия: архивирование электронных документов выполняется для их хранения в течение неопределенно долгого периода времени, тогда как резервное копирование данных выполняется в выбранные моменты для ограниченного по времени хранения. Тем не менее часто резервное копирование называют архивированием или созданием архивов, поэтому в данной статье эти термины будут использованы как синонимы.

Ограничение объема архивных хранилищ данных заставляет искать компромисс между частотой резервного копирования данных и сроком их хранения. Резервные копии обычно имеют высокую избыточность и содержат низкий процент изменений в последовательных версиях. При создании файловых архивов процедуры резервного копирования используют сочетание полных и инкрементальных архивов, когда в инкрементальный архив попадают только недавно измененные файлы или измененные фрагменты файлов.

Другим методом экономного хранения данных является поиск с отождествлением общих фрагментов разных версий файлов, называемый дедупликацией данных. В этом случае архивы не связаны между собой, как в цепочке инкрементальных архивов, и поэтому могут одинаково эффективно добавляться, распаковываться и удаляться, независимо от того, как давно они были созданы. За такую гибкость часто приходится расплачиваться сложностью работы алгоритма дедупликации и непрозрачностью схемы хранения уникальных фрагментов данных, найденных этим алгоритмом.

Распространение технологий виртуализации и контейнеризации увеличило плотность нагрузки в дата-центрах и повысило необходимость наличия инструментов для централизованного и эффективного резервного копирования данных. Однако для большинства распространенных систем виртуализации такие инструменты доступны пока только в виде дополнительных коммерческих программ и сервисов. Важным исключением является Proxmox Backup Server [3]. Эта программа с открытым исходным кодом, которая имеет функции дедупликации и сжатия архивов, а также обеспечивает тесную интеграцию со средой виртуализации Proxmox VE, что позволяет с помощью функции QEMU dirty bitmaps автоматически отслеживать моди-

фикацию данных и копировать только измененные блоки виртуальных дисков. Когда доступны лишь базовые функции среды виртуализации, они, тем не менее, позволяют создавать снимки состояний виртуальных систем и затем преобразовывать их в цепочки инкрементальных архивов (например, см. [2], где описан такой метод хранения резервных копий виртуальных ОС в среде Libvirt/QEMU/KVM).

Если же не рассматривать системы резервного копирования, интегрированные с прикладным ПО и средами виртуализации, то остается несколько способов оптимального хранения архивных данных:

1. LVM, ZFS, BTRFS). Недостаток этого способа состоит в сложности управления Использование моментальных снимков общего архивного хранилища средствами менеджера томов или файловой системы (архивом, т.к. в нем одновременно хранятся разные объекты данных, возможно, требующие разных расписаний резервного копирования.

2. Размещение архива в файловой системе со встроенной функцией дедупликации данных, например, ZFS или BTRFS. Данный способ кажется простым и удобным, но он не совместим со сжатием данных перед внесением в архив, а встроенные в файловую систему алгоритмы сжатия малоэффективны, так как применяются к блокам малых размеров. Кроме того, применение дедупликации в этих файловых системах снижает быстродействие ввода-вывода и потребляет значительный объем оперативной памяти.

3. Использование универсальных утилит архивирования файлов, имеющих функции дедупликации и компрессии данных.

Поскольку последний метод оказывается наиболее простым, эффективным и гибким, именно для него в НИИСИ РАН авторами была разработана утилита HIGGS, использование которой запланировано для оптимизации процесса резервного копирования в среде виртуализации Libvirt/QEMU/KVM вместо текущего метода цепочек инкрементальных архивов [2].

2. Алгоритм работы программы HIGGS

Архиватор HIGGS является программой на языке командного интерпретатора Bash, которая использует утилиту GNU parallel [1] для параллельной обработки данных, утилиту sha1sum для вычисления контрольных сумм по алгоритму SHA1 и стандартные утилиты сжатия данных (gzip, bzip2, xz, zstd). При архивировании файла утилита GNU parallel разбивает его данные на последовательность блоков, которые передаются набору параллельных процессов для анализа, сжатия и записи в архив. Каждый из этих процессов обрабатывает переданный ему блок данных независимо от других процессов и возвращает строку с описанием полного имени файла сжатого и сохраненного блока, которую утилита GNU parallel передает головному процессу для создания индексного файла. Важно, что эта утилита отслеживает соответствие порядка блоков данных и результатов работы подпроцессов, для чего служит опция --keep-order. Необходимо отметить, что операция сжатия и записи блока данных должна быть ограждена блокировками, иначе в случае совпадения хэш-сумм разных блоков возникнет ситуация гонки (race condition) между параллельными процессами с неопределенным результатом. В качестве идентификатора блокировок используется хэш-сумма блока, поэтому задержки возникают только в случае повторений во входных данных, когда выигрыш от дедупликации компенсирует время, потерянное на ожидание доступа.

На рис. 1 приведен пример хранения двух файлов, имеющих общие блоки данных: vol1 и vol2. Например, эти файлы могут быть архивами последовательных состояний виртуальной ОС. Архивированные данные хранятся одновременно в центральном хранилище, и у пользователей, где архив каждого файла является отдельным каталогом.

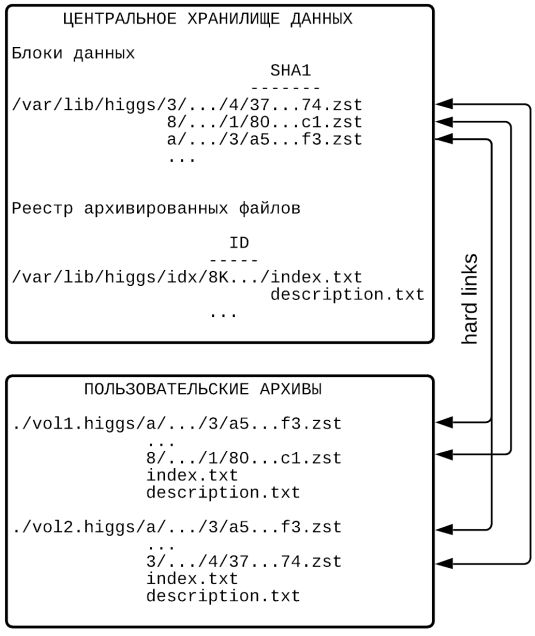

Рис. 1. Устройство хранения данных архиватором HIGGS

Имена сохраняемых блоков данных соответ-

ствуют SHA1 контрольным суммам их содержимого. Каждый блок сжимается утилитой компрессии данных (на рис. 1 в примере использована утилита zstd) и хранится в единственном экземпляре, что обеспечивается функцией создания ссылок типа hard links в файловых системах Unix/GNU Linux. Пользовательский каталог архива кроме блоков данных содержит файл (index.txt), перечисляющий блоки данных, и файл метаданных (description.txt), которые необходимы для распаковки файла и восстановления его атрибутов. Центральное хранилище также присваивает каждому архиву уникальный идентификатор (ID) и содержит копии индексных файлов и файлов с метаданными, что позволяет восстанавливать данные при случайном удалении пользователем его архивного каталога.

В процессе извлечения файла из архива утилита GNU parallel передает строки индексного файла процессам распаковки, которые независимо находят файлы блоков и выполняют декомпрессию данных. Затем эта же утилита собирает в нужном порядке полученные фрагменты данных и записывает их в выходной файл.

От размера блоков данных, на которые разбивается архивируемый файл, зависят степень компрессии данных и накладные расходы на создание процессов обработки блоков данных. Увеличение размеров блоков ускоряет работу, но также повышает расход дискового пространства, если области модифицированных данных значительно меньше размера блоков. Предельным случаем таких изменений служит модификация одного байта, требующая хранения новой версии всего блока. Чтобы избежать подобной неэффективности, утилита HIGGS использует алгоритм рекурсивного разбиения блоков. Разбиение на две равные части выполняется для исходного блока данных, но продолжается для каждого из полученных блоков, только если на предыдущей итерации одна из частей блока оказалась уже сохраненной ранее.

Работу этого алгоритма иллюстрирует рис. 2, где приведен пример последовательного архивирования исходного блока данных и пяти его модификаций. На этом рисунке белым цветом обозначены блоки, содержащие исходные или измененные данные, т.е. те данные, которые требуется записать. Прямоугольники с пунктирной границей обозначают ранее сохраненные блоки, для них архиватор создает только файловые ссылки. Блоки с серой заливкой обозначают сегменты данных, содержимое которых не было изменено, но которые требуют записи, поскольку их хэш-сумма еще не известна архиватору. Хранение этих блоков можно считать вынужденным, однако они могут быть использованы, если возникнут другие изменения в данных блока. Например, при сохранении модификации 5 используются блоки, созданные для модификаций 1 и 2.

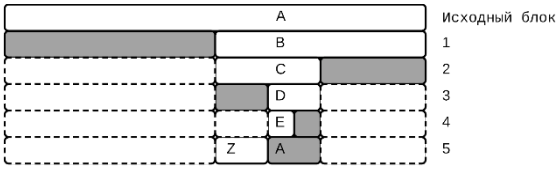

Рис. 2. Пример работы алгоритма рекурсивного разбиения блока

При дальнейших изменениях в позиции A-B-C-D-E, например, при переборе всех остальных значений этого байта, на диске будет занято место, приблизительно соответствующее трем блокам, вместо размера в 256 блоков, как было бы при отсутствии разбиения. Применение алгоритма разбиения блоков не требует коммуникации между параллельными процессами, запускаемыми утилитой GNU parallel, и не усложняет остальные части алгоритма работы HIGGS.

3. Алгоритм работы программы HIGGS

Для оценки скорости работы и степени сжатия данных утилита HIGGS тестировалась вместе с несколькими общедоступными утилитами резервного копирования файлов: Borg v.1.1.17 [4], Bupstash v.0.10.2 [5], Rdedup v.3.2.1 [6] и ZBackup v.1.5-alpha-32-g3e4977c [7]. Все эти утилиты также имеют функции дедупликации и сжатия данных и распространяются вместе с исходными кодами с лицензиями, разрешающими их свободное использование и модификацию. В таблице 1 приведены результаты сравнения производительности HIGGS и четырех вышеупомянутых утилит.

Тестирование выполнялось на сервере Intel Xeon E-2186G, 4x32Gb Micron DDR4 ECC UDIMM 2666MHz, работающем под управлением ОС GNU Linux CentOS 8 с версией ядра v.4.18.0-240.10.1.el8_3.x86_64. В процессе тестирования все файлы располагались в оперативной памяти, в файловой системе RAM-диска, созданного командой `mount -t ramfs -o size=32G /mnt/ramfs`. Такой режим был выбран, чтобы избежать зависимости результатов от особенностей работы конкретных моделей SSD и HDD дисков и определить максимально достижимую скорость работы тестируемых инструментов. Тест состоял в последовательном архивировании двух снимков виртуального диска системы GNU Linux Lubuntu 21.10. Файлы виртуальных дисков не имели специального формата (т.е. использовался raw-формат). Первый снимок был сделан непосредственно по-

сле установки, а второй — после установки нескольких программных пакетов общим объемом 1.7Гб. При одинаковом номинальном размере этих файлов, равном 10Гб, общие размеры ненулевых данных составляли 5.9Гб и 7.6Гб соответственно. Размеры этих файлов в сжатом виде (zstd,3) были равны 2.10Гб и 2.86Гб, соответственно, а размер изменений между этими файлами, вычисленных с помощью команды `xdelta3 -S djw -5 -s`, составлял 895Мб. Архиватор HIGGS использовал размер блока 16Мб, в остальных программах применялись настройки по умолчанию. Каждый тест повторялся 3 раза и в качестве результата записывалось наименьшее время его выполнения.

Таблица 1. Сравнение производительности пяти инструментов резервного копирования данных. Выделенные жирным шрифтом значения обозначают два наилучших результата для каждого из тестов

Инструмент	Режим компрессии	Версия файла	Архивирование, сек	Извлечение, сек	Объем архива, Мб
Borg	zstd,3	1	56.8	29.4	**2114336**
		2	42.1	31.1	**3074804**
Bupstash	lz4	1	24.8	**10.3**	3005184
		2	15.7	**10.8**	4180452
HIGGS	zstd,2	1	**10.8**	**16.9**	2215076
		2	**8.5**	**16.8**	3365004
Rdedup	zstd,3	1	**15.4**	21.8	2268428
		2	**8.7**	23.4	3099604
ZBackup	lzma,6	1	330.2	121.6	**1608608**
		2	165.7	151.9	**2252300**

Из таблицы 1 видно, что HIGGS является лидером по скорости архивирования, при этом упаковывая данные до объема, сходного с объемом архивов Borg и Rdedup. Для Borg, Rdedup и ZBackup режим и степень компрессии подбирались из соображений разумного компромисса между скоростью работы и объемом архива: выбирался наиболее качественный алгоритм из реализованных в программе, а уровень компрессии повышался, пока не наступало резкое снижение скорости работы. При этом ZBackup показывает наибольшую степень компрессии данных, однако в нем реализован только алгоритм lzma, который работает значительно медленнее остальных алгоритмов. Архиватор Bupstash достигает наибольшей скорости извлечения данных, но при этом создает архив наибольшего размера – в 1.5-2 раза больше, чем остальные инструменты. Причина в том, что LZ4, единственный реализованный в Bupstash алгоритм, отличается высокой скоростью распаковки и низким средним уровнем компрессии данных.

4. Обсуждение

Результаты тестирования подтвердили высокую скорость работы алгоритма HIGGS, превышающую значения, показанные конкурирующими продуктами. Вместе с тем по степени сжатия архивов HIGGS является одним из лидеров. Этот результат достигается несмотря на то, что HIGGS — это скрипт на языке командного интерпретатора Bash, тогда как остальные программы написаны на компилируемых языках программирования (C/C++/Rust). Алгоритм HIGGS не может распознать сдвиг данных, возникающий в результате вставки или удаления фрагментов файлов. Известны алгоритмы точного поиска изменений в данных, обычно они используют вычисление контрольной суммы в скользящем окне (rolling hash, rolling checksum). Однако применение скользящего окна трудно совмещается с параллельной обработкой блоков данных, поэтому утилита HIGGS использует рекурсивное разбиение блоков для оптимизации хранения локальных изменений данных. Следует отметить, что вставки и удаления в больших файлах редко встречаются в реальных прикладных программах. Так, содержимое виртуальных дисков имеет внутреннюю структуру файловой системы, в которой данные файлов собраны из блоков и не сдвигаются при обычных операциях. Содержимое томов, где хранятся данные СУБД, также имеет внутреннюю структуру, оптимизированную для быстрого доступа к данным, поэтому в большинстве операций измененные данные оказываются расположены локально. Можно предположить, что общие требования оптимального использования накопителей данных стимулируют применять схемы хранения, использующие перезапись блоков данных вместо вставок и удалений. Редкими исключениями из этого правила служит запись больших объемов данных в XML-файлах и вывод дампов

баз данных в текстовом формате. Такие практики неэффективны и вне контекста архивирования, поэтому, как правило, могут быть заменены хранением данных в бинарных форматах, имеющих встроенную структуру.

4.1. Преимущества программы HIGGS

Устройством программы HIGGS объясняются ее следующие положительные качества:

Простота. Схема хранения данных проста и наглядна, что обеспечивает надежность и способствует пониманию устройства архиватора HIGGS.

Гибкость. Метод и степень сжатия данных могут быть выбраны для каждой команды, помещающей файл в хранилище. Размер блоков данных также может быть изменен, например, чтобы лучше соответствовать гранулярности изменяемых данных.

Универсальность. Утилита HIGGS разработана в среде GNU Linux, но может быть использована в любой Unix-подобной ОС, где установлены утилиты Bash и GNU parallel, а также реализована поддержка hard links в файловой системе.

Скорость. Работа HIGGS хорошо распараллеливается на многоядерных процессорах, что обеспечивается утилитой GNU parallel. Кроме того, алгоритм архиватора не выполняет лишних записей на диск: блок данных записывается только в том случае, если это необходимо и когда он уже сжат. Это ускоряет работу и экономит ресурс записи SSD-накопителей.

Надежность. Случайное удаление папки архива не приводит к потере данных, так как центральное хранилище HIGGS содержит ссылки на те же блоки данных. Там же продублированы индексные файлы и файлы метаданных. И наоборот, удаление центрального хранилища оставляет нетронутыми архивы пользователей. Это следствие схемы хранения данных HIGGS и автоматического отслеживания числа ссылок на файлы блоков данных.

4.2. Ограничения программы HIGGS

Простота устройства программы HIGGS приводит и к некоторым ограничениям области ее применения:

Локальность хранилища. Использовать HIGGS можно только в пределах одной файловой системы, которая должна поддерживать функцию подсчета числа ссылок на файл (hard links).

Трудности создания внешних резервных копий. Средства для резервного копирования файловой системы на внешние хранилища данных должны понимать устройство файловых ссылок (hard links). Иначе получившиеся архивы будут неоправданно велики, а вся экономия от дедупликации аннулируется. Избежать этой проблемы позволит копирование только общего хранилища HIGGS, где все блоки данных хранятся в одном экземпляре. Впоследствии из этого хранилища можно извлечь файлы средствами HIGGS так же, как и в случае случайного удаления пользователем его файлов.

Ограничение производительности. В случае применения очень быстрых накопителей информации (например, SSD-устройств с интерфейсом NVMe) и малым размером блоков данных (<4Mb) видно, что утилита GNU parallel и командный интерпретатор Bash значительно нагружают процессор. Кроме того, становится видна нагрузка от процессов ядра ОС, отвечающих за подсистему блочных устройств. Эту проблему можно решить ценой небольшого увеличения объема архива, используя блоки большего размера.

4.3. Возможные улучшения программы HIGGS

Программа HIGGS может быть дополнена следующими функциями:

Усиленная защита от искажения данных. Для обнаружения и исправления скрытых искажений данных (silent data corruption) могут быть применены алгоритмы избыточного кодирования. Например, программы Parchive или PAR2 [8] используют коды Рида-Соломона для восстановления данных с неисправного носителя.

Псевдофайловая система. Драйвер FUSE (File system in USErspace) для монтирования архива в виде псевдофайловой системы мог бы упростить доступ к архивным данным. Например, с помощью такой функции можно было бы непосредственно подключать архивные файлы для запуска временных виртуальных ОС с состоянием виртуального диска на момент архивации. Из рассмотренных выше утилит только программа Borg имеет данную функцию.

5. Заключение

Построенная на базе готовых инструментов GNU Linux программа архивирования данных HIGGS продемонстрировала высокую производительность и эффективность. Тестовая эксплуатация этой программы показала существенное сокращение объема архивного хранилища и ускорение процесса создания резервных копий виртуальных ОС. Кроме практического значения программы HIGGS, она служит примером того, что с помощью утилиты GNU parallel можно создавать высокопроизводительные приложения для параллельной обработки данных.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН

(Проведение фундаментальных научных исследований (47 ГП) по теме № FNEF-2021-0007 «Развитие методов математического моделирования распределенных систем и соответствующих методов вычисления. 0580-2021-0007», Рег. № 121031300051-3).

HIGGS – Backup Tool with Data Deduplication and Compression

A.B. Betelin, I.B. Egorychev, A.A. Prilipko, G.A. Prilipko, S.G. Romanyuk, D.V. Samborskiy

Abstract. Virtualization environments and database management systems need backing up large amount of unstructured data in order to recover from a fatal hardware failure or accidental data loss. Low rate of data changes results in a high degree of data redundancy unless incremental backups or data deduplication are performed. This paper presents the algorithm of a universal archiver, Hard link-Inspired Granular Gracious Storage (HIGGS), specially designed for optimal backup storage of arbitrary binary data. Benchmarks have shown HIGGS's superiority over several popular backup tools that also perform data deduplication.

Keywords: backup, deduplication, GNU parallel, hard link, data compression

Литература

1. O. Tange. GNU Parallel the Command Line Power Tool. login: The USENIX Magazine 36, 1 (Feb 2011), 42–47.
2. А.Б. Бетелин, А.А. Прилипко, Г.А. Прилипко, С.Г. Романюк, Д.В. Самборский. Оптимизация архивирования виртуальных операционных систем в среде GNU Linux/QEMU/KVM/Libvirt, «Труды НИИСИ РАН», Том 8 (2018), № 5, С. 32-41. ISSN 2225-7349.
3. Сайт проекта Proxmox Backup Server, https://www.proxmox.com/en/proxmox-backup-server. Дата обращения 15.10.2021.
4. Сайт проекта Borgbackup, https://www.borgbackup.org. Дата обращения 15.10.2021.
5. Сайт проекта Bupstash, https://bupstash.io. Дата обращения 15.10.2021.
6. Сайт проекта Rdedup, https://github.com/dpc/rdedup. Дата обращения 15.10.2021.
7. Сайт проекта ZBackup, http://zbackup.org. Дата обращения 15.10.2021.
8. Страница википедии Parchive, http://en.wikipedia.org/wiki. Дата обращения 15.10.2021.

Ускорение быстрого преобразования Фурье на основе технологии OpenCL

А.А. Бурцев[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, burtsev@niisi.msk.ru

Аннотация. Статья посвящена применению технологии OpenCL, позволяющей использовать мощные ресурсы графических процессоров для повышения быстродействия вычислительных программ. Рассматриваются приёмы разработки в среде OpenCL эффективных параллельных программ для ускорения операции быстрого преобразования Фурье.

Ключевые слова: параллельное программирование, технология OpenCL, гетерогенные системы, микропроцессоры семейства КОМДИВ, быстрое преобразование Фурье

1. Введение

Почти все современные высокопроизводительные системы для ускорения вычислений опираются на тот или иной вид параллельной обработки. Предполагается, конечно, что преобразованная в параллельную форму программа решения вычислительной задачи будет исполняться не одним, а сразу несколькими процессорами. И потому ожидается, что в результате совместной параллельной работы многих процессоров удастся добиться значительного ускорения решения вычислительной задачи.

Но чтобы добиться ожидаемого ускорения, необходимо уметь преобразовать алгоритм решения вычислительной задачи к параллельному виду, т.е. разбивать программу на такие участки, которые могут исполняться разными процессорами параллельно (в одно и то же время).

Для преобразования обычных вычислительных программ в параллельные уже предложен целый ряд самых разнообразных технологий. Так, для создания параллельной программы, призванной исполняться на разных процессорах (процессорных ядрах) одного компьютера с общей памятью, используется технология OpenMP [1, п. 5.1] [2, гл.3]. А для построения распределённой программы, исполнение которой предполагается на семействе (кластере) компьютеров, связанных сетью, применяется технология MPI [1, п. 5.2] [2, гл.4].

Для ускорения вычислений в рамках одного компьютера предлагаются также специализированные сопроцессоры SIMD класса. В микропроцессорном семействе КОМДИВ [3], разрабатываемом в ФНЦ НИИСИ РАН, к таким относятся векторный сопроцессор CPV и сопроцессор цифровой обработки сигналов CP2. Такие математические спецпроцессоры, действительно, позволяют в несколько раз повысить производительность определённого класса вычислительных задач: главным образом, задач линейной алгебры [4] и цифровой обработки сигналов [5,6].

Недавно было открыто ещё одно направление исследований, обещающее значительное ускорение вычислительной программы за счёт её особого распараллеливания. Это направление предполагает использование в вычислительных программах ресурсов мощных графических процессоров. Такие специализированные процессоры первоначально встраивались в видеокарты (или графические адаптеры), управляющие формированием видеосигнала, поступающего на монитор компьютера, и применялись там для ускорения цифровой обработки изображений и видеоинформации. Почти каждая современная видеокарта содержит целый массив (десятки и даже сотни) однородных специализированных процессоров, которые можно ухитриться использовать как слаженный ансамбль параллельно функционирующих исполнителей (вычислителей) для ускоренного решения одной прикладной задачи.

Применять графические карты как ускорители вычислений первыми предложили специалисты компании NVIDIA, разработав (для своих видеокарт) технологию **CUDA** (**C**ompute **U**nified **D**evice **A**rchitecture). Затем аналогичную технологию, но уже пригодную для видеокарт различных семейств, предложила компания Apple. А Khronos Group как организация, занимающаяся выработкой открытых стандартов, приняла эту технологию за основу и довела её до стандарта, дав ей название **OpenCL** (**Open C**omputing **L**anguage) [7].

В настоящее время OpenCL служит стандартом разработки параллельных программ для так называемых гетерогенных (неоднородных) систем, в которых наряду с обычными универсаль-

ными процессорами (CPU) для исполнения программы могут использоваться специализированные процессоры самого разнообразного характера, в том числе графические (GPU), а также процессоры обработки сигналов (DSP).

Благодаря технологии OpenCL сегодня даже самый обычный настольный компьютер, снабжённый достаточно мощной видеокартой, может быть использован как общедоступный инструмент для ускорения вычислительных задач, как своеобразный суперкомпьютер «под рукой» на рабочем столе инженера или учёного.

В новую микросхему семейства КОМДИВ, разрабатываемого в ФНЦ НИИСИ РАН, включёно графическое ядро с поддержкой технологии OpenCL (версии 1.2). А такое ядро можно использовать не только для ускоренной обработки видеоинформации, но и для ускорения вычислительных задач самого разнообразного характера.

Для испытания возможностей графического ядра этой микросхемы в НИИСИ на основе технологии OpenCL был разработан ряд прикладных программ обработки векторов и матриц, характерных для задач линейной алгебры и цифровой обработки сигналов. Полученные в результате прогонов этих программ показатели производительности свидетельствуют о том, что с помощью технологии OpenCL можно получить значительный выигрыш в ускорении решения таких вычислительных задач, в которых обрабатываются большие массивы данных.

Данная статья продолжает знакомство с технологией OpenCL, начатое автором в [8], где рассматривались приёмы разработки эффективных программ по технологии OpenCL для решения некоторых типичных задач линейной алгебры (главным образом, операции перемножения больших матриц). Теперь предлагается уделить внимание способам построения OpenCL-программ для ускорения операции быстрого преобразования Фурье (БПФ), являющейся одной из ключевых в задачах обработки сигналов.

Изложение материала статьи сопровождается таблицами, в которых представлены количественные показатели ускорения операции БПФ, полученные в результате прогонов разработанных OpenCL-программ на различных OpenCL-платформах. Эти показатели характеризуют, какие коэффициенты ускорения операции БПФ удаётся получить в среде OpenCL на обычных (настольных) компьютерах семейства Intel; и какое ускорение возможно получить в среде OpenCL, поддерживаемой имеющейся в настоящее время микросхемой графического ускорителя семейства КОМДИВ.

2. Основные алгоритмы для преобразования Фурье

Во многих задачах цифровой обработки сигналов применяется операция дискретного преобразования Фурье (**ДПФ** или **DFT** – **D**iscrete **F**ourier **T**ransform). Рассмотрим сначала алгоритмы и формулы, по которым осуществляется такое преобразование в обычных программах, рассчитанных на последовательное их исполнение одним процессором.

2.1. Формулы и алгоритм ДПФ

Операция ДПФ над заданным вектором комплексных чисел Y_N заключается в получении результирующего вектора X_N комплексных чисел, элементы которого X_k вычисляются на основе исходного вектора Y по правилу:

$$X_k = \sum_{j=0}^{N-1} \left\{ Y_j \times W_N^{k \cdot j} \right\}$$

Здесь используются так называемые весовые коэффициенты вида W^q_N, которые вычисляются как комплексные величины по формуле:

$$W_N^q = e^{-i \cdot 2\pi \cdot q/N},$$

которую можно выразить иначе:

$$W_N^q = \cos\frac{2\pi q}{N} - i \cdot \sin\frac{2\pi q}{N},$$

применяя формулу Эйлера:

$$e^{i \cdot \varphi} = \cos\varphi + i \cdot \sin\varphi,$$

где i – мнимая комплексная единица.

Чтобы при исполнении ДПФ каждый раз не вычислять весовые коэффициенты заново, их можно посчитать заранее и приготовить в виде таблицы – массива комплексных чисел W (размером от 0 до N). Тогда вычисление ДПФ непосредственно по этому правилу можно выразить следующим алгоритмом (функцией на языке C):

```
void DFT(int N, complex *X,
         complex *Y, complex *W)
{ int k,j; complex Xk;
 for (k=0; k<N; k++) {
  Xk= 0.0 + I*0.0;
  for (j=0; j<N; j++) {
   Xk= Xk+Y[j]*W[(k*j)%N];
  }//for j
  X[k]= Xk;
 }//for k
}//DFT
```

Хотя этот алгоритм и выполняет требуемое преобразование (ДПФ), он редко применяется на практике, по крайней мере, для преобразований комплексных векторов большого размера. Главный его недостаток заключается в том, что он

требует слишком длительного времени исполнения (даже с заранее подготовленной таблицей весовых коэффициентов). Дело в том, что в этом алгоритме для получения результирующего вектора длины N приходится исполнять порядка $O(N^2)$ операций с комплексными числами. Вот почему такой алгоритм характеризуется вычислительной сложностью $O(N^2)$.

2.2. Базовый алгоритм БПФ

Существует, однако, целый ряд алгоритмов, которые позволяют выполнить ДПФ гораздо быстрее. Среди них выделяется особый класс алгоритмов под общим названием «быстрое преобразование Фурье» (**БПФ** или **FFT** – **F**ast **F**ourier **T**ransform), которые характеризуются вычислительной сложностью $O(N\log_2 N)$.

Далее в качестве базового алгоритма для исполнения преобразования Фурье на одном процессоре (последовательной программой) примем вариант известного алгоритма Кули-Тьюки с прореживанием по времени для векторов комплексных чисел длиной $N=2^P$ (степени двойки). Математическое обоснование такого алгоритма можно найти, например, в [6], а его описание подробно изложено в [9, гл.12], а также и в [5]. Здесь лишь поясним схематично суть этого алгоритма так, чтобы в дальнейшем стала понятной его трансформация в параллельную программу для OpenCL-среды.

Рассматриваемый алгоритм БПФ состоит их двух частей. В первой части с элементами заданного вектора выполняется так называемая «битреверсная» перестановка. В ходе такой перестановки элемент вектора с индексом **k** меняется местами с элементом, стоящим в векторе на позиции **m**, целочисленное значение которого получается записью в обратном порядке двоичного значения k длиной $L=\log_2 N$: если $\mathbf{k}=(b_L,b_{L-1},\ldots,_2,b_1)$, то $\mathbf{m}=(b_1,b_2,\ldots,b_{L-1},b_L)$.

Вторая часть БПФ проводится в несколько этапов ($P=\log_2 N$). На каждом из них все элементы вектора (по определённому правилу) разбиваются на пары, и над каждой из них выполняется особая операция комплексной арифметики, получившая образное название «бабочка Фурье» (**БФ** или **BF** – **B**utterFly of **F**ourier).

На 1-ом этапе пары, над которыми предстоит исполнить операции БФ, образуются из соседних элементов вектора. На 2-ом этапе они составляются из элементов, отстоящих друг от друга на расстоянии 2-х позиций. На каждом последующем этапе расстояние s между образуемыми парами удваивается (s=1,2,..,N/2). Так что на очередном t-ом этапе (t=1,2,..,P) величина s вычисляется по формуле $s=2^{(t-1)}$, а пары для исполнения операций БФ образуются из элементов вектора с индексами [a] и [a+s].

Операция «бабочка Фурье» **BF(A,B,V)** над комплексными величинами **A** и **B** с коэффициентом **V** предписывает вычислить новые значения A′ и B′ для этих величин на основе их исходных значений по правилу:

$A' = A + B\times V,\ B' = A - B\times V$

Обычно при программной реализации такой операции сначала вычисляют произведение B×V как вспомогательную величину T, а за тем уже, используя её, вычисляют новые значения величин A и B:

T= B×V; B= A – T; A= A + T;

Оформим такую реализацию операции БФ в форме макроопределения на языке Си:

```
#define BF(A,B,V,T) \
 { T=B*V; B=A-T; A=A+T; }
```

Замечание. Это макроопределение задаёт действия с комплексными величинами, и предполагается, что используемые в нём операции сложения, умножения и вычитания применяются к параметрам (A,B,V,T) встроенного типа **complex**. Если же используется версия языка Си, в котором нет такого встроенного типа данных, то тело такого макроопределения придётся, конечно же, реализовать по-другому.

В OpenCL-программе, обогащающей (включением директивы «#include **<CL/cl.h>**») язык Си новыми характерными для OpenCL-среды возможностями, для реализации операций с комплексными величинами можно использовать встроенный тип **cl_float2** и приготовить соответствующие функции:

```
#define complex cl_float2
complex cAdd(complex X, complex Y)
{ complex V; V.s[0]=X.s[0]+Y.s[0];
 V.s[1]=X.s[1]+Y.s[1]; return V; }
complex cSub(complex X, complex Y)
{ complex V; V.s[0]=X.s[0]-Y.s[0];
 V.s[1]=X.s[1]-Y.s[1]; return V; }
complex cMul(complex X, complex Y)
{ complex V;
 V.s[0]=X.s[0]*Y.s[0]-X.s[1]*Y.s[1];
 V.s[1]=X.s[0]*Y.s[1]+X.s[1]*Y.s[0];
 return V;}
```

И уже с их помощью реализовать макроопределение операции БФ:

```
#define BF(A,B,V,T)\
{ T=cMul(B,V); \
  B=cSub(A,T); A=cAdd(A,T); }
```

Конец замечания.

Теперь представим описанный алгоритм БПФ в виде функции на языке Си:

```
void FFT(int N, complex *X,
 complex *Y,complex *W)

{long int P,t,s,h,G,R,d,k,u,j,a,b;

 complex V,T;
```

```
// Часть 1. Бит-реверсная перестановка:
 BRVPRST(N,X,Y);
//ч2.Основной цикл исполнения бабочек Фурье:
 P=iLog2(N); // P= log2N (так что N=2^P)
 s=1;h=2;G=1;R=N/2;d=N/2;
 for(t=1;t<=P;t++) { // на t-ом этапе:
  for(k=0,u=0;k<G;k++,u=u+d){
   V=W[u];
   for(j=0,a=k;j<R;j++,a=a+h)
   {b=a+s; BF(X[a],X[b],V,T);}
  }//for k
  h=h*2;s=s*2;G=G*2;R=R/2;d=d/2;
 }//for t
}//FFT
```

Для более детального пояснения алгоритма этой функции приведём краткую характеристику используемых в нём величин:

P – кол-во этапов (итераций)=$\log_2 N$, ($N=2^P$)

t – номер очередного этапа (итерации) t=1,2,..,P

На каждом t-ом этапе:

G – количество групп =$2^{(t-1)}$ = 1,2,..,N/2

k – номер очередной группы =0,1,..,G-1

R – число пар в каждой группе =$2^{(P-t)}$ = N/2,..2,1; так что G*R=N/2 (количество всех пар)

j – номер очередной пары в группе =0,1,..,R-1

s – расстояние между элементами пары =$2^{(t-1)}$

h – расстояние между парами = 2^t =2*s

a,b – индексы элементов j-ой пары в k-ой группе: a= k+j*h; b = a+s;

d – множитель индекса для доступа в таблицу W весовых коэффициентов =$2^{(P-t)}$, (d=R)

u – индекс доступа в таблицу весовых коэффициентов W для всех пар k-ой группы = k*d

Функция **FFT** принимает исходный вектор **Y** длины **N** и формирует результат преобразования Фурье как новый вектор комплексных чисел **X**, используя для вычислений приготовленную таблицу коэффициентов **W**. Сначала она вызывает процедуру **BRVPRST** для формирования в векторе **X** бит-реверсной перестановки вектора **Y**, а затем уже приступает к многократным вычислениям «бабочек Фурье» над всевозможными парами элементов вектора **X**.

На очередном шаге цикла по t (t-ом этапе) обрабатываемые пары элементов распределяются по группам так, чтобы в каждой группе обрабатывались те пары, для которых операция БФ выполняется с одним и тем же весовым коэффициентом. На каждом шаге цикла по k для k-ой группы вычисляется индекс u для получения из таблицы W их общего весового коэффициента V=W[u], с которым далее выполняются операции БФ для всех пар этой группы, перебираемых в цикле по j.

Такое распределение пар на группы можно наглядно изобразить, представив обрабатываемый на t-ом этапе вектор X как матрицу из 2R строк и G столбцов, уложенную в том же самом месте памяти по строкам (см. рис.1). Каждый столбец этой матрицы будет представлять одну группу, а соседние элементы в нём образуют R пар для исполнения операций БФ.

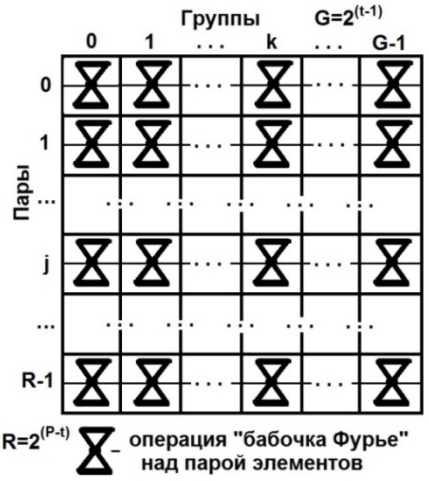

Рис. 1. Группы пар для вычислений бабочек Фурье

На каждом этапе алгоритма БПФ образуется N/2 пар элементов, и над каждой парой выполняется операция комплексной арифметики «бабочка Фурье». Всего исполняется P=$\log_2 N$ этапов, поэтому вычислительная сложность такого алгоритма оценивается величиной O(N$\log_2 N$).

В специализированных сопроцессорах сигнальной обработки (см. [10, гл.3]) операция БФ, как правило, реализуется одной аппаратной командой, за счёт чего, главным образом, и обеспечивается значительное ускорение исполнения на них алгоритма БПФ. Например, такая команда предусмотрена в сопроцессорах CPV (cmaddsub) и CP2 (buttertfly) микропроцессорного семейства КОМДИВ. А в многопроцессорной среде

OpenCL добиться ускорения представленного алгоритма БПФ можно путём его преобразования в программу, рассчитанную на параллельное исполнение.

3. Первоначальный вариант OpenCL-программы для БПФ

Чтобы построить программу по технологии OpenCL (или OpenCL-программу), способную функционировать в OpenCL-среде, содержащей множество параллельных исполнителей, необходимо создать основную программу (OpenCL-приложение на языке Си) и несколько так называемых процедур ядра (на языке OpenCL). Основная программа запускается на хосте (основном процессоре). Она подготавливает OpenCL-среду и периодически запускает в ней приготовленные процедуры ядра параллельно сразу на всех её вычислительных узлах, так называемых обрабатывающих элементах (ОЭ или PE – processing element).

3.1. Процедуры OpenCL-ядра для распараллеливания алгоритма БПФ

Рассмотрим, как можно ускорить алгоритм БПФ, рассчитывая на наличие нескольких параллельных исполнителей в среде OpenCL.

При выполнении первой части БПФ требуется осуществить бит-реверсную перестановку заданного вектора Y_N (или получить новый вектор X_N). Если длина вектора N, то в нём образуется N/2 пар. Элементы каждой пары можно переставлять независимо от других пар. Значит, такую работу могут параллельно выполнить N/2 исполнителей, если её распределить так, чтобы каждый из них переставил только одну назначенную ему уникальную пару. (Заметим, что при m=k пару (X_k,X_m) можно и не переставлять вовсе).

Если же требуется осуществить не перестановку, а бит-реверсное копирование, т.е. получить новый вектор X_N в качестве бит-реверсной перестановки заданного вектора Y_N, тогда каждый элемент Y на позиции k надо записать в вектор X на позицию m, целочисленное значение которой получить как перевёрнутое двоичное значение индекса k. Опять же, запись каждого элемента Y_k в X_m можно производить независимо от записи других элементов, поэтому можно ускорить такое бит-реверсное копирование вектора Y в вектор X, поручив осуществить его N параллельным исполнителям.

Для этого параллельного бит-реверсного копирования приготовим такую процедуру ядра:

```
__kernel void fft_brvprst
( const uint L,
  __global complex *X,
  __global const complex *Y,
  __global const uint *BRT)
{ uint k,m;
  k= get_global_id(0);
  m= BRT[k]; // m=бит-реверсное значение k
X[m]= Y[k];
}//fft_brvprst
```

Заметим, что операция получения бит-реверсного значения **m** длины **L** для заданного целого числа **k** (типа unsigned long int) не является такой уж тривиальной. Для её реализации может потребоваться исполнить значительное количество машинных команд. Вот как, например, такую операцию можно было бы записать операторами языка Си:

```
m=k;
m=m&0x55555555)<<1|(m&0xaaaaaaaa)>>1;
m=m&0x33333333)<<2|(m&0xcccccccc)>>2;
m=m&0x0f0f0f0f)<<4|(m&0xf0f0f0f0)>>4;
m=m&0x00ff00ff)<<8|(m&0xff00ff00)>>8;
m=m&0x0000ffff)<<16|(m&0xffff0000)>>16;
m=m>>(32-L);
```

Подобный громоздкий блок операторов пришлось бы поместить в тело процедуры ядра fft_brvprst вместо оператора m=BRT[k]. Однако, чтобы при каждом вызове процедуры ядра не вычислять каждый раз заново бит-реверсное значение индекса, можно приготовить заранее таблицу, в которой каждому целочисленному значению k=0,1,..,N-1 сопоставить его бит-реверсное значение длины **L** (L=$\log_2$N), и передавать эту таблицу в качестве параметра (BRT), что в действительности и обеспечивается при вызове процедуры fft_brvprst.

Вторая часть алгоритма БПФ состоит из P этапов (P=$\log_2$N). На каждом этапе из элементов вектора образуется N/2 пар, над которыми выполняется операция БФ. Такая операция над каждой парой может выполняться независимо от других пар. А значит, над всеми парами операцию БФ можно выполнять параллельно, выделяя для такой работы на каждом этапе N/2 параллельных исполнителей (обрабатывающих элементов в среде OpenCL).

Для осуществления на очередном **t**-ом этапе такого параллельного исполнения всех операций БФ над элементами вектора составим следующую процедуру ядра:

```
__kernel void fft_btfly
(int n,__global complex *X,
  __global complex *W, int t)
{ uint i,G,R,k,j,s,h,a,b,u;
  i= get_global_id(0);
  G=1<<(t-1); // кол-во групп G= 2^(t-1)
  R=n>>t; // кол-во пар в группе R= 2^(P-t);
  s=G; h=1<<t;
  k=i&(G-1); // номер группы = i % G
  j=i>>(t-1);// номер пары в группе= i / G
  s=G; h=2*s;
```

```
 a=k+j*h; b=a+s; u=R*k;
 BTF(X[a],X[b],W[u]);
}//fft_btfly
```

Такая процедура будет запускаться на каждом обрабатывающем элементе (см. далее вызов функции **clEnqueueNDRangeKernel** на **t**-ом шаге цикла в теле функции clFFT). Используемые в ней вспомогательные величины имеют тот же смысл, что и в рассмотренном ранее алгоритме **FFT**, но их значения вычисляются немного иначе.

Каждый исполнитель, вызвав функцию **get_global_id**(0), сначала узнаёт назначенный ему уникальный номер (i=0,1,..,N/2-1). Используя параметр t как номер этапа, вычисляет по формулам: $G=2^{(t-1)}$ и $R=2^{(P-t)}$, сколько на данном этапе образуется групп G и сколько пар R в каждой группе. Затем определяет, какую пару из всех имеющихся N/2 пар ему надлежит обрабатывать: из какой она должна быть группы (k) и какая она в ней по порядку (j). Распределение пар по исполнителям осуществляется (см. рис.1) по формулам: **k= i mod G**, **j= i div G**.

Далее в теле процедуры ядра вычисляются индексы (**a**,**b**) элементов выбранной пары, а также индекс **u** для взятия весового коэффициента из таблицы, заданной параметром **W**. И наконец, над назначенной парой исполняется операция «бабочка Фурье», которая в теле процедуры ядра реализована в виде вызова макроса **BTF(**X[a],X[b],W[u]**)**.

А само макроопределение для операции БФ комплексной арифметики запишем с учётом того, что в нём для представления комплексных величин (типа **complex**) будем использовать встроенный тип данных float2 языка OpenCL. Такой тип данных задаёт пару вещественных значений одинарной точности, разрешая исполнять над ними попарно операции вещественной арифметики (сложения, вычитания, умножения), и допуская обращение к его реальной и мнимой частям, как к полям структуры {**.x**,**.y**}:

```
#define complex float2
#define BTF(XA,XB,V) \
{complex Xa,Xb,XbV; \
 Xa=XA; Xb=XB; /* XbV=Xb×V) : */  \
 XbV.x= Xb.x*V.x - Xb.y*V.y; \
 XbV.y= Xb.x*V.y + Xb.y*V.x; \
 XA= Xa + XbV; XB= Xa - XbV; }
```

Сосредоточим описанные процедуры ядра вместе с сопровождающими их вспомогательными функциями и макроопределениями в файле “fft.cl”. Строки этого файла будут анализироваться при компоновке программного кода ядра. (Это осуществляется далее путём вызова функции clCreateProgramWithSource в основной программе).

3.2. Основная программа для запуска исполнения БПФ в среде OpenCL

Допустим, что в нашей основной программе уже проделаны все типовые подготовительные действия по инициализации OpenCL-среды, которые требуется обеспечить для всякой OpenCL-программы. Будем считать, что уже определена OpenCL-платформа, получены дескрипторы OpenCL-устройств, подготовлен OpenCL-контекст **cntxt** (типа cl_context**)** и в нём создан дескриптор очереди команд **cmndQ** (типа cl_command_queue).

А также должным образом приготовлены все объекты **objX**, **objY**, **objW, objBRT** (типа cl_mem) для представления в памяти OpenCL-данных, подлежащих обработке, и особый объект **prgrm** (типа cl_program), содержащий в откомпилированной форме код всех процедур ядра:

```
prgrm=clCreateProgramWithSource(cntxt,1,
  (const char**)&cSourceString, NULL, NULL);
```

Как обеспечить все эти действия, было подробно описано автором в пп. 2.3, 2.4, 2.5 в [8].

Каждой (описанной ранее) процедуре ядра сопоставим свой дескриптор (типа cl_kernel), вызвав функцию **clCreateKernel** и задав в ней (2-м параметром) имя процедуры:

```
cl_kernel kn1=
  clCreateKernel (prgrm, "fft_brvprst", NULL);
cl_kernel kn2=
  clCreateKernel (prgrm, "fft_btfly", NULL);
```

Для выполнения БПФ в среде OpenCL оформим отдельную функцию с заголовком:

```
void clFFT(int N, complex *X,

 complex *Y)
```

И предусмотрим в её теле следующую последовательность действий.

1. Загрузим вектор Y в буфер объекта **objY**:

```
cl_uint szC= sizeof(complex);

clEnqueueWriteBuffer(cmndQ,

  objY,CL_TRUE,0,szC*N,Y,0,0,0);
```

2. Приготовим процедуру ядра fft_brvprst к запуску, назначив ей 4 фактических параметра:

```
cl_uint P=iLog2(N);

cl_uint szI= sizeof(cl_uint);

cl_uint szM= sizeof(cl_mem);

clSetKernelArg(kn1,0,szI,&P);

clSetKernelArg(kn1,1,szM,&objX);

clSetKernelArg(kn1,2,szM,&objY);
```

```
clSetKernelArg(kn1,3,szM,&objBRT);
```

3. Запустим в OpenCL-среде процедуру ядра fft_brvprst на множестве из N исполнителей (обрабатывающих элементов) для параллельного бит-реверсного копирования вектора из буфера объекта **objY** в буфер объекта **objX:**

```
size_t gWS[1]= {N};

cl_event ev1;

clEnqueueNDRangeKernel(cmndQ,

 kn1,1,NULL,gWS,NULL,0,NULL,&ev1);
```

И дождёмся окончания её завершения (всеми исполнителями):

```
clFinish(cmndQ);
```

4. Подготовим к многократному запуску процедуру ядра fft_btfly, установив для неё первые 3 фактических параметра:

```
clSetKernelArg(kn2,0,szI,&N);

clSetKernelArg(kn2,1,szM,&objX);

clSetKernelArg(kn2,2,szM,&objW);
```

Последний её параметр (4-ый), задающий номер этапа (t), будет назначаться позднее на каждом очередном шаге цикла по t.

5. Теперь зададим основной цикл (по t), на каждом шаге t которого процедура ядра fft_btfly запускается на множестве из N/2 исполнителей для параллельного выполнения всех операций БФ t-го этапа (t=1,..,P):

```
cl_uint t; gWS[0]=N/2;

cl_event ev2;

for (t=1; t<=P; t++) {

 clSetKernelArg(kn2,3,szI,&t);

 clEnqueueNDRangeKernel(cmndQ,

 kn2,1,NULL,gWS,NULL,0,NULL,&ev2);

 clWaitForEvents(1,&ev2);

}//for t
```

Перед окончанием каждого шага цикла следует дождаться завершения вызванной процедуры ядра всеми исполнителями, для чего и вызывается функция **clWaitForEvents**.

6. Наконец, полученный в буфере объекта **objX** результирующий вектор запишем в X:

```
clEnqueueReadBuffer(cmndQ,

 objX,CL_TRUE,0,szC*N,X,0,0,0);
```

Основная программа OpenCL-приложения сначала вызывает функцию FFT для обычного (последовательного) исполнения БПФ, а затем подготавливает OpenCL-среду и вызывает в ней функцию clFFT для выполнения той же задачи, но уже в среде параллельных исполнителей. Прогоны этих функций можно выполнять многократно (по нескольку раз), что позволяет в результате получить усреднённые показатели времени их исполнения.

Замечание. Предполагается, что используемые в этих функциях таблицы (бит-реверсных индексов BRT и весовых коэффициентов Фурье W) приготавливаются единожды перед последующим многократным вызовом этих функций. Конец замечания.

3.3. Результаты ускорения БПФ в OpenCL-среде платформы Intel

Попробуем теперь оценить, насколько удалось ускорить операцию БПФ с помощью OpenCL-программы только что рассмотренного варианта. Такой вариант (A) программы первоначально был разработан (на языке Си) как консольное приложение в среде MS Visual Studio 2017. И был скомпонован для исполнения под ОС Windows-10 в среде OpenCL для платформы с аппаратной конфигурацией, содержащей основной универсальный процессор CPU Intel-i59400 (с частотой 2.9 Ггц) и встроенный графический процессор GPU UHD 630 (с частотой 350 МГц). (Далее будем называть её сокращённо «платформа Intel»). Скомпонованная программа многократно прогонялась для векторов комплексных чисел (одинарной точности) различной длины $N=2^P$ (P=8,9,..,24). Результаты этих прогонов представлены в таблице 1.

В этой таблице (как и в последующих далее таблицах) приведены следующие показатели:

1) $\mathbf{T_{CPU}}$ – время исполнения функции FFT на одном процессоре (CPU);

2) $\mathbf{T_{CL}}$ – полное время исполнения функции clFFT в уже подготовленной OpenCL-среде;

3) $\mathbf{T_{Я}}$ – «чистое» совокупное время непосредственного исполнения операции БПФ процедурами ядра параллельно всеми обрабатывающими элементами OpenCL-среды;

4) **К1** – общий коэффициент ускорения операции БПФ, обеспечиваемый данной программой в среде OpenCL, который вычисляется как отношение времени исполнения $\mathbf{T_{CPU}}$ операции БПФ на CPU к полному времени $\mathbf{T_{CL}}$ исполнения этой же операции в среде OpenCL;

5) **К2** – «чистый» коэффициент ускорения операции БПФ, обеспечиваемый данной программой в среде OpenCL, который вычисляется как отношение времени исполнения $\mathbf{T_{CPU}}$ операции БПФ на CPU к времени исполнения $\mathbf{T_{Я}}$ этой же операции непосредственно процедурами ядра на

вычислительных узлах среды OpenCL без учёта исполнения необходимых подготовительных действий на хосте (основном процессоре).

Таблица 1. Ускорение операции БПФ OpenCL-программой варианта А на платформе Intel

P	$N=2^P$	T_{CPU}	T_{CL}	$T_{Я}$	K1	K2
8	256	0.066629	0.800043	0.030496	0.083	2.185
9	512	0.153393	0.874386	0.034830	0.175	4.404
10	1024	0.341213	0.961005	0.040414	0.355	8.443
11	2048	0.753087	1.062318	0.047663	0.709	15.800
12	**4096**	1.663742	1.173253	0.060329	**1.418**	27.578
13	8192	3.678552	1.313642	0.08541	2.800	43.069
14	16384	7.80999	1.464376	0.131244	5.333	59.507
15	**32768**	17.01297	1.71237	0.21491	9.935	**79.162**
16	65536	40.0665	2.1941	0.5181	18.26	77.34
17	131072	85.6315	3.0949	1.16016	27.67	67.78
18	262144	185.2239	5.3368	2.7328	**34.71**	67.78
19	524288	404.0427	11.8328	7.6057	34.15	53.12
20	1048576	1005.2896	29.8731	20.8819	33.65	48.14
21	2097152	2290.5496	75.7643	63.2002	30.23	36.24
22	4194304	5182.2881	179.7021	153.1009	28.84	33.85
23	8388608	11036.229	342.0574	312.8234	32.26	35.28
24	16777216	23724.545	725.0618	658.2147	32.72	36.04

T_{CPU} – время исполнения на CPU
T_{CL} – полное время исполнения в OpenCL
$T_{Я}$ – время исполнения OpenCL-ядра
(все времена даны в миллисекундах)
K1 – общий коэффициент ускорения = T_{CPU} / T_{CL}
K2 – «чистый» коэффициент ускорения = T_{CPU} / $T_{Я}$

Из этой таблицы видно, что применение технологии OpenCL на этой платформе позволяет ускорить операцию БПФ над векторами длиной N>=4096. Судя по коэффициенту К1, удаётся добиться весьма значительного общего ускорения (почти в **30** раз) для векторов очень большого размера (длины 2^{18} и более) А если принимать во внимание «чистый» коэффициент К2, то можно сказать, что в некоторых случаях (см. для $N=2^{15}$) удаётся добиться почти **80**-кратного ускорения исполнения операции БПФ ядрами OpenCL-устройства.

3.4. Результаты ускорения БПФ в OpenCL-среде платформы NVidia

В действительности, с применением технологии OpenCL можно добиться ещё большего ускорения вычислительных операций, если подключить к компьютеру более мощную специализированную видеокарту.

Разработанный вариант OpenCL-программы преобразования Фурье был опробован и для другой OpenCL-платформы, аппаратная конфигурация которой помимо универсального процессора Intel i3-2100 (на частоте 3.1 Ггц) содержит подключённую к нему специализированную видеокарту NVidia GeForce 1050ti (с частотой 1392 Мгц). (Будем называть её сокращённо «платформа NVidia»).

OpenCL-программа рассмотренного выше варианта (А) многократно прогонялась на этой платформе для векторов комплексных чисел (одинарной точности) различной длины $N=2^P$ (P=8,9,..,24). Полученные в результате таких прогонов усреднённые показатели ускорения операции БПФ представлены в таблице 2.

Таблица 2. Ускорение операции БПФ OpenCL-программой варианта А на платформе NVidia

P	$N=2^P$	T_{CPU}	T_{CL}	$T_{Я}$	K1	K2
8	256	0.150000	0.850001	0.033856	0.176	4.431
9	512	0.300000	0.850001	0.038464	0.353	7.800
10	1024	0.600001	1.000001	0.037536	0.600	15.985
11	2048	1.400002	0.700001	0.046080	**2.000**	30.382
12	4096	3.000004	1.100002	0.046816	2.727	64.081
13	8192	6.600010	1.00000	0.053376	6.60	123.65
14	16384	14.400022	1.400002	0.074592	10.29	193.05
15	32768	30.800041	1.400002	0.104608	22.00	294.43
16	65536	68.500099	1.500005	0.165184	45.67	**414.69**
17	131072	146.00020	3.00000	0.522240	48.67	279.57
18	262144	313.00046	4.00001	1.284736	78.25	243.63
19	524288	829.00116	7.00001	2.768096	118.4	299.48
20	1048576	1725.0023	12.00002	5.907296	143.8	292.01
21	2097152	3764.0054	24.00002	12.59418	156.8	298.87
22	4194304	8166.4111	48.00006	26.93549	170.1	303.18
23	8388608	17470.225	98.00015	57.80224	178.3	302.24
24	16777216	37264.453	196.000	123.1616	**190.1**	**302.6**

T_{CPU} – время исполнения на CPU
T_{CL} – полное время исполнения в OpenCL
Тя – время исполнения OpenCL-ядра
(все времена даны в миллисекундах)
K1 – общий коэффициент ускорения = **T_{CPU} / T_{CL}**
K2 – «чистый» коэффициент ускорения = **T_{CPU} / Тя**

Из неё видно, что на этой платформе удаётся получить более значительные результаты ускорения: коэффициент общего ускорения K1 достигает значения **190** (на векторе длиной 2^{24}), а коэффициент «чистого» ускорения K2 достигает значения **414** (при $N=2^{16}$). Заметим также, что почти что на всех векторах большого размера ($\geq 2^{15}$) отмечается примерно трёхсоткратное (**300**!) «чистое» ускорение операции БПФ.

3.5. Результаты ускорения БПФ в OpenCL-среде платформы КОМДИВ

Представим теперь, какие же коэффициенты ускорения операции БПФ можно получить в среде OpenCL, обеспечиваемой в настоящее время микросхемой графического ускорителя семейства КОМДИВ. Описанный выше вариант OpenCL-программы испытывался на аппаратной платформе, обеспечиваемой этой микросхемой, со следующими частотами: частота основного процессора (CPU) 400 Мгц, частота памяти 800 Мгц, частота системной шины 600 Мгц и частота графического процессора (GPU) 200 Мгц. (Будем называть далее такую платформу сокращённо «платформа КОМДИВ»)

OpenCL-программа многократно прогонялась на такой платформе для векторов комплексных чисел (одинарной точности) различной длины $N=2^P$ (P=8,9,..,23). (Для векторов длины 2^{24} операцию БПФ на этой платформе не удавалось исполнить из-за ограниченного объёма памяти OpenCL-устройства). Полученные в результате этих прогонов усреднённые коэффициенты ускорения операции БПФ на платформе КОМДИВ представлены в итоговой таблице 3.

Таблица 3. Ускорение операции БПФ OpenCL-программой варианта А на платформе КОМДИВ

P	$N=2^P$	T_{CPU}	T_{CL}	Тя	K1	K2
8	256	0.894785	22.79573	10.0040	0.039	0.089
9	512	2.000136	25.07366	11.1940	0.080	0.179
10	1024	4.419073	27.44511	12.4550	0.161	0.355
11	2048	9.677856	29.96424	13.4220	0.323	0.721
12	4096	21.1156	32.1007	14.283	0.658	**1.478**
13	8192	46.0372	33.19956	15.0040	**1.387**	3.068
14	16384	100.1620	38.7069	17.004	2.588	5.890
15	32768	257.6530	52.45069	24.429	4.912	10.547
16	65536	602.8322	80.5334	39.4420	7.485	15.284
17	131072	1352.147	132.3809	69.2160	10.214	19.535
18	262144	2921.8594	256.4551	145.1360	11.393	20.132
19	524288	6337.456	510.0666	306.832	12.425	20.654
20	1048576	13553.918	1037.776	652.848	13.061	**20.761**
21	2097152	28616.545	2158.878	1407.207	13.255	20.336
22	4194304	60443.551	4503.303	3020.369	13.422	20.012
23	8388608	127307.38	9449.790	6506.731	**13.472**	19.565

T_{CPU} – время исполнения на CPU
T_{CL} – полное время исполнения в OpenCL
Тя – время исполнения OpenCL-ядра
(все времена даны в миллисекундах)
K1 – общий коэффициент ускорения = **T_{CPU} / T_{CL}**
K2 – «чистый» коэффициент ускорения = **T_{CPU} / Тя**

Конечно, представленные в этой таблице коэффициенты ускорения выглядят достаточно скромными по сравнению с предыдущими двумя OpenCL-платформами, функционирование которых обеспечивается несколькими мощными графическими ядрами (compute units). Но ведь и получены они на минимально возможной аппаратной конфигурации (всего лишь с одним графическим ядром), обеспечивающей поддержку технологии OpenCL (версии 1.2) в микропроцессорах семейства КОМДИВ.

Можно сравнить приведённые в таблице 3 показатели ускорения операции БПФ для комплексных векторов одинарной точности с коэффициентами ускорения той же операции, которые были получены на векторном сопроцессоре CPV семейства КОМДИВ. Полученные на CPV коэффициенты ускорения приведены в таблице 7 в [5]. Заметим, что согласно этой таблице на векторном сопроцессоре обеспечивается ускорение операции БПФ в 10-11 раз для всех векторов комплексных чисел одинарной точности длиной N=64,..,4096, которые целиком помещаются в кэш-памяти на весь период времени исполнения этой операции. Однако, для векторов, не помещающихся целиком в кэш-памяти, коэффициент ускорения (KV) операции БПФ на CPV резко снижается.

В результате такого сравнения можно утверждать, что с помощью применения технологии OpenCL на графическом ускорителе удаётся добиться таких же высоких коэффициентов ускорения операции БПФ для векторов больших размеров ($N>2^{16}$), которые векторный сопроцессор позволял обеспечивать лишь для векторов малых размеров ($<=2^{12}$).

4. Варианты развития OpenCL-программы для ускорения БПФ

Рассмотренный (в п.3) первоначальный вариант (A) OpenCL-программы имеет ряд недостатков, избавившись от которых, можно рассчитывать получить более совершенные версии программы для ускорения операции БПФ. Эти недостатки обусловлены тем, что представленный вариант OpenCL-программы не использует в полной мере все возможности, предоставляемые средой OpenCL.

Прежде всего, заметим, что запускаемые этой программой процедуры ядра не извлекают никакой выгоды из иерархического строения памяти OpenCL-устройства. Они слабо используют самую быстродействующую внутреннюю (private) память самих обрабатывающих элементов (processing elements) и совсем не используют локальную (local) память рабочей группы (workgroup), в которую можно объединять несколько обрабатывающих элементов для совместной работы. В результате каждый параллельный исполнитель вынужден всякий раз обращаться за данными в общую глобальную память, что, конечно же, замедляет их совместную работу.

К тому же для исполнения второй части БПФ (см. в п.3.2 5-й пункт списка действий при описании функции clFFT) организуется цикл, в котором на каждом шаге (этапе) процедура ядра fft_btfly запускается заново на всех рабочих элементах (work-items). А это, в известной степени, увеличивает накладные расходы, затрачиваемые на запуск параллельных исполнителей и синхронизацию их совместной работы.

Были опробованы несколько возможных путей усовершенствования представленной выше OpenCL-программы, направленных на устранение отмеченных недостатков. В процессе такого усовершенствования были разработаны новые варианты такой программы, которые используют рабочие группы и предоставляемую ими локальную память, а также активно эксплуатируют внутреннюю память самих обрабатывающих элементов. Применённые в этих вариантах приёмы оптимизации позволяют немного усилить коэффициенты ускорения операции БПФ, но лишь для векторов очень больших размеров.

К сожалению, достигается такое усиление ценой весьма значительного усложнения программного кода. Так что не представляется возможным дать исчерпывающее объяснение всех этих приёмов оптимизации в рамках одной данной статьи.

5. Заключение

Таким образом, технологию OpenCL можно успешно применять для ускорения процесса исполнения программ таких вычислительных задач, при решении которых требуется осуществлять однотипные операции над огромными массивами данных.

В частности, с помощью применения технологии OpenCL можно реально ускорить выполнение операции быстрого преобразования Фурье для векторов комплексных чисел большого размера. Причём, можно добиться ускорения в десятки и даже сотни раз, если поддерживающая такую технологию аппаратная платформа способна обеспечить одновременное слаженное функционирование огромного массива параллельных вычислительных узлов.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН «Проведение фундаментальных научных исследований (47 ГП)» по теме № FNEF-2021-0004 «Разработка архитектуры, системных решений и методов для создания микропроцессорных ядер и коммуникационных средств семейства систем на кристалле двойного назначения. 0580-2021-0004», Рег. № 121031300049-0.

Acceleration of Fast Fourier Transform Based on OpenCL Technology

A.A. Burtsev

Abstract. The article is devoted to the use of OpenCL technology, which allows using the powerful resources of graphic proces-sors to increase the speed of computing programs. Development techniques of efficient parallel programs in the OpenCL environment to accelerate the fast Fourier transform operation are considered.

Keywords: parallel programming, OpenCL technology, heterogeneous systems, microprocessors of the KOMDIV family, fast Fourier transform

Литература

1. В.В. Воеводин, Вл.В. Воеводин. Параллельные вычисления. Спб., БХВ-Петербург, 2004.

2. С.В. Борзунов, С.Д. Кургалин, А.В. Флегель. Практикум по параллельному программированию. Спб., БХВ, 2017.

3. Официальный сайт ФНЦ НИИСИ РАН. Разработка СБИС. Развитие микропроцессоров с архитектурой КОМДИВ, https://www.niisi.ru/devel.htm

4. А.А. Бурцев. О возможности оптимизации некоторых функций библиотеки линейной алгебры с помощью векторного сопроцессора. «Труды НИИСИ РАН», Т. 4 (2014), № 2, 5–15.

5. А.А. Бурцев. О возможности применения векторного сопроцессора для ускорения операции быстрого преобразования Фурье. «Труды НИИСИ РАН», Т. 5 (2015), № 2, 138–147.

6. О.Ю. Сударева. Эффективная реализация алгоритмов быстрого преобразования Фурье и свёртки на микропроцессоре КОМДИВ128-РИО. М., НИИСИ РАН, 2014.

7. Официальный OpenCL–сайт организации Khronos Group, http://www.khronos.org/opencl/

8. А.А. Бурцев. Оптимизация операции перемножения матриц на основе технологии OpenCL. «Труды НИИСИ РАН», Т. 10 (2020), № 5-6, 100–112.

9. Стивен Смит. Цифровая обработка сигналов. Практическое руководство для инженеров и научных работников. М., Додека-XXI, 2012.

10. В.В. Корнеев, А.В. Киселёв. Современные микропроцессоры. М., НОЛИДЖ, 2000.

Описание метода построения библиотеки отображения растровой карты, инвариантной к форматам целевых геопространственных данных

П.В. Егоров[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, egorov@niisi.ras.ru, +7(903)5524887.

Аннотация. Рассматривается метод построения библиотеки «tilelib», инвариантной к форматам целевых пространственных данных. Функции библиотеки «tilelib» предназначены для отображения цифровой растровой карты с помощью устройств вывода графической информации. Метод рассматривается на примере целевых данных, соответствующих спецификации MBTiles. Метод является частью технологии построения изображения растровой электронной карты с использованием программных средств для операционной системы реального времени, разработанных в ФГУ ФНЦ НИИСИ РАН.

Ключевые слова: растровая электронная карта, тайловая карта, тайл, спецификация MBTiles, библиотека «tilelib»

Введение

Целью данной работы является описание метода построения библиотеки отображения растровой карты, инвариантной к форматам целевых геопространственных данных (далее инвариантная библиотека), в программной среде, разработанной в ФГУ ФНЦ НИИСИ РАН.

В настоящее время имеется несколько десятков форматов растровых электронных карт [1]. В связи с этим становится актуальной задача создания программ, инвариантных к форматам целевых геопространственных данных (далее инвариантная программа).

Одним из способов решения этой задачи является метод, который состоит в том, что для программы создается собственная модель данных растровой карты, в которую преобразуется модель целевой карты. Преобразование осуществляется с помощью программ конверторов, которые реализуются для каждого формата целевых данных. В результате инвариантная программа независимо от формата целевых данных будет оперировать данными во внутреннем формате.

Такой подход к созданию инвариантных программ применяется в библиотеке абстракции геопространственных данных (Geospatial Data Abstraction Library - GDAL). GDAL - это библиотека-конвертор для форматов растровых и векторных геопространственных данных, выпускаемая Open Source Geospatial Foundation с открытым исходным кодом.

Библиотека обеспечивает программам единые абстрактные модели растровых и векторных данных [2]. Целевые данные перед выполнением операций с ними с помощью подпрограмм преобразуются к внутреннему формату данных библиотеки.

Метод построения инвариантной библиотеки является частью технологии построения изображения растровой электронной карты, которая была рассмотрена в работе [3].

В основу этой технологии были положены следующие принципы.

1. Построение изображения растровой электронной карты с помощью функций библиотеки «tilelib».

2. Обеспечение инвариантности библиотеки «tilelib» к форматам целевых геопространственных данных.

3. Выполнение графических операций с помощью программных изделий графическая библиотека (ГБ), графический сервер (ГС) и библиотека интерфейсных компонентов (БИК), разработанных в ФГУ ФНЦ НИИСИ РАН.

4. Исполнение программ на ЭВМ семейства «Багет» под управлением операционной системы реального времени (ОС РВ).

В работе [3] был рассмотрен вопрос построения изображения растровой электронной карты с помощью библиотеки «tilelib».

В данной статье будет рассмотрен вопрос обеспечения инвариантности библиотеки «tilelib» к целевым данным.

Метод разработан для целевых растровых карт имеющих тайловую модель данных. В указанной модели изображение строится из матриц пикселей, которые называются тайлами [3].

Электронную растровую карту с тайловой моделью данных далее будем называть тайловой картой.

Для тестирования библиотеки «tilelib» была реализована контрольная задача.

В контрольной задаче в качестве целевых данных использовались данные формата MBTiles.

MBTiles - это спецификация схемы базы данных SQLite для хранения данных тайловой карты [4].

При реализации инвариантной библиотеки «tilelib» применялся следующий подход:

1. Для библиотеки «tilelib» была разработана модель данных тайловой карты.

2. Для формата целевых данных MBTiles была реализована программа (далее драйвер), которая преобразует структуру целевых данных к формату данных библиотеки «tilelib».

3. В программе драйвере для тайловой карты в формате MBTiles экспортировались функции библиотек для языка Си «libpng» и СУБД «SQLite» с открытым исходным кодом.

В данной работе моделирование статических и динамических аспектов программной архитектуры библиотеки «tilelib» осуществляется с помощью методов абстрагирования, формализации, прямого и обратного проектирования, алгоритмизации и тестирования.

В качестве средства формализации используется язык моделирования UML.

Перечислим основные принципы метода построения инвариантной библиотеки «tilelib»:

- Создание для библиотеки «tilelib» модели данных тайловой карты.

- Преобразование структуры целевых данных к формату данных библиотеки «tilelib» с помощью программы драйвера.

- Применение целевых электронных растровых карт с тайловой моделью данных.

- Использование среды программирования, разработанной в ФГУ ФНЦ НИИСИ РАН.

1. Спецификация MBTiles

В спецификации MBTiles допускается применение графики тайлов в растровом и векторном виде.

Так как данные в векторном формате не обрабатываются библиотекой «tilelib», в этом разделе будет рассматриваться спецификация MBTiles только для растровой графики.

В таблице 1 приводится описание таблиц базы данных спецификации MBTiles.

Таблица 1. Таблицы базы данных

Имя таблицы	**Описание**
metadata	Данные настроек
tiles	Данные тайлов

Описание атрибутов таблицы metadata приводится в таблице 2.

Таблица 2. Структура таблицы metadata

Атрибут	**Описание**
name	Имя параметра
value	Значение параметра

Для каждого параметра настроек создается строка в таблице metadata.

Параметры в спецификации версии 1.3 делятся на три группы: обязательные, рекомендуемые и необязательные. Описания каждой группы параметров приводятся в таблицах 3, 4 и 5.

Таблица 3. Обязательные параметры

Значение атрибута «name»	**Описание**
name	Задает имя набора тайлов
format	Определяет формат данных тайла

В спецификации MBTiles определены три типа форматов растровых данных: jpg, png и webp. Кроме того, данный список типов форматов можно расширить с помощью IETF типов.

Таблица 4 Рекомендуемые параметры

Значение атрибута «name»	**Описание**
bounds	Максимальный размер отображаемой области карты. Границы должны определять область, охватываемую всеми уровнями масштабирования. Границы представлены в виде значений широты и долготы точек углов прямоугольника растра левого нижнего и правого верхнего.
center	Долгота, широта центральной точки растра и уровень масштабирования для отображения карты по умолчанию
minzoom	Минимальный уровень масштабирования электронной карты
maxzoom	Максимальный уровень масштабирования электронной карты

Таблица 5. Необязательные параметры

Значение атрибута «name»	**Описание**
attribution	Описание источника данных и стиля карты
description	Описание данных электронной карты. - type - тип данных overlay или baselayer. - version - версия электронной карты.

Атрибуты таблицы tiles перечислены в таблице 6.

Таблица 6. Структура таблицы tiles

Атрибут	**Описание**
zoom_level	Масштабный уровень
tile_column	Столбец таблицы тайлов
tile_row	Строка таблицы тайлов
tile_data	Данные тайла в двоичном виде в растровом формате, тип которого задается параметром «format»

Для каждого тайла в таблице tiles создается отдельная запись.

Назначение атрибутов zoom_level, tile_column и tile_row определяется спецификацией «Tile Map Service Specification» (далее TMSS) [5]

Масштабный уровень zoom_level задает разрешение отображаемой карты, которое определяет количество единиц измерения координат в одном пикселе карты и зависит от типа выбранной проекции.

Логическую структуру данных тайловой карты можно представить как таблицу столбцов и строк на пересечении которых находятся ячейки с тайлами. Тогда атрибут tile_column будет задавать номер колонки тайла, а tile_row — номер строки тайла.

В спецификации TMSS нумерация колонок и строк на каждом уровне zoom_level начинается с нуля, что соответствует координатам точки нижнего левого угла растра карты в десятичных градусах (180 °W, 85.0511 °S).

2. Архитектура инвариантной библиотеки «tilelib»

Архитектура библиотеки «tilelib» показана на рисунке 1 в виде диаграммы пакетов.

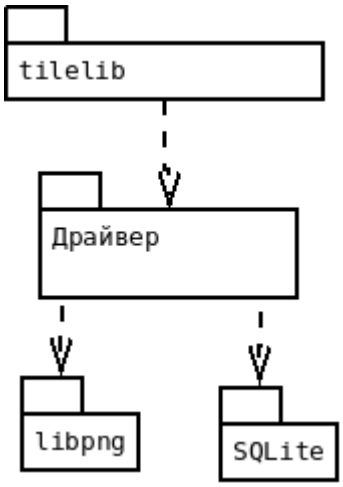

Рис. 1. Архитектура инвариантной библиотеки «tilelib»

В данной статье будет использоваться словарь языка UML для обозначения сущностей предметной области, в том числе термины пакет, класс, объект, контейнер, операция, метод.

Перечисленные термины имеют следующие значения.

Пакет - это механизм организации элементов диаграммы в группы. Другое возможное определение пакета - подмножество множества элементов модели.

Класс - это описание множества объектов с одинаковыми атрибутами, операциями, связями и семантикой [6]. Альтернативное определение класса - множество однотипных объектов.

Объектом (экземпляром) называется конкретное воплощение абстракции, к которому могут быть применены операции и которое обладает состоянием, сохраняющим их результаты [6].

Операция - это сервис, который может быть запрошен у любого объекта класса для реализации поведения [6].

Метод - это реализация операции объектом. С помощью вызовов методов осуществляется передача сообщений объекту.

Ассоциация агрегирования - это такое отношение между классами, при котором объекты одного класса являются частью объектов другого класса.

Ассоциация композитного агрегирования - это ассоциация агрегирования, в которой «объект-часть может принадлежать только единственному целому, и, кроме того, как правило,

жизненный цикл частей совпадает с циклом целого» [7].

Контейнер – объект, предназначенный для хранения других объектов и предоставляющий операции для доступа или итерации по содержащимся в нем элементам [6]. Термин контейнер будет так же использоваться для классов, которые описывают соответствующие объекты.

Далее в тексте и на диаграммах для обозначения объекта перед именем класса объекта будет ставиться двоеточие, а само название выделяться *курсивом*. В случае необходимости именовать объект перед двоеточием будет указываться имя объекта [3].

На диаграмме показаны пакеты «tilelib», «Драйвер», «libpng» и «SQLite».

Кооперации объектов перечисленных выше пакетов имеют следующее функциональное назначение.

Пакет «tilelib» содержит классы, объекты которых выполняют отображение тайловой карты на графическом устройстве. Пакет «tilelib» является абстракцией библиотеки «tilelib».

Пакет «Драйвер» группирует классы объектов, реализующие преобразование формата целевых данных в формат данных пакета «tilelib». Объекты классов пакета «tilelib» передают сообщения объектам классов пакета «Драйвер», инициируя процедуру преобразования данных. На диаграмме сообщения между пакетами «tilelib» и «Драйвер» показаны с помощью пиктограммы отношения зависимости (пунктирная стрелка). В этом отношении пакет «tilelib» зависит от пакета «Драйвер».

В данной работе в качестве целевых данных инвариантной библиотеки рассматриваются данные в формате «MBTiles». В соответствии со спецификацией данного формата целевые данные размещаются в БД SQLite. Операции с базой данных обеспечивают объекты классов пакета «SQLite», который является абстракцией библиотеки SQLite. На диаграмме с помощью пиктограммы отношения зависимости показаны вызовы объектами класса «Драйвер» методов объектов классов пакета «SQLite». В этом отношении пакет «Драйвер» зависит от пакета «SQLite».

В спецификации «MBTiles» определено несколько растровых форматов тайла, в данной статье рассматривается один из них - формат PNG.

Для преобразования данных в формате PNG в формат XImage, который является форматом данных объектов класса «Тайл» пакета «tilelib», объекты классов пакета «Драйвер» передают сообщения объектам классов пакета «libpng», которые в свою очередь выполняют операции с данными в формате PNG. На диаграмме сообщения между пакетами «Драйвер» и «libpng» обозначены пиктограммой отношения зависимости. В этом отношении пакет «Драйвер» зависит от пакета «libpng». Пакет «libpng» является абстракцией библиотеки «libpng».

3. Модель данных пакета «tilelib»

На рисунке 2 модель данных пакета «tilelib» показана в виде диаграммы классов.

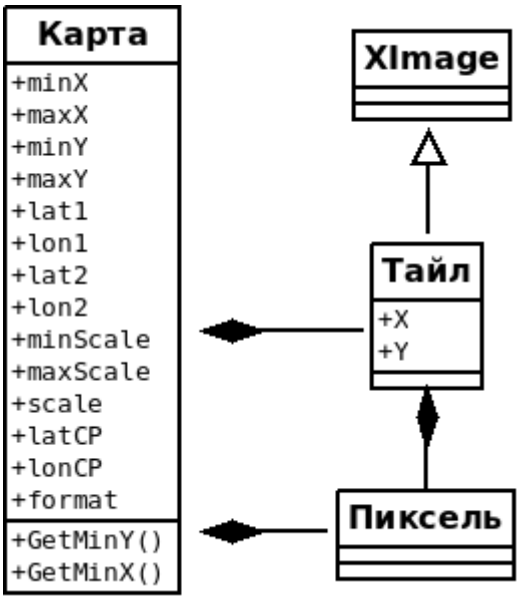

Рис. 2. Диаграмма классов пакета «tilelib»

Классы «Карта», «Тайл» и «Пиксель» описывают соответственно данные тайловой карты, тайла и пикселя и операции с ними [3].

Класс «Карта» является контейнером для класса «Тайл» и «Пиксель». Это отношение между классами, в том числе означает, что операции класса «Карта» могут выполняться как с объектами класса «Тайл», так и с объектами класса «Пиксель».

Класс «Тайл» является контейнером для класса «Пиксель», что следует из определения тайла как матрицы пикселей.

Указанные отношения между классами «Карта», «Тайл» и «Пиксель» на диаграмме показаны с помощью пиктограмм ассоциации композитного агрегирования (стрелка с закрашенным ромбом на конце).

В пакете «tilelib» класс «Тайл» наследует атрибуты и операции от класса «XImage» пакета «ГБ» (с помощью пакета ГБ моделируется программное изделие графическая библиотека, описание пакета «ГБ» см. [3]). Далее объекты класса «Тайл» пакета «tilelib» будем обозначать как «*Ximage:Тайл*».

В модели данных пакета «tilelib» данные контейнера «*tilelib*:*Карта*» логически организованы в виде таблицы столбцов и строк на пересечении которых находятся ячейки с объектами «*Ximage:Тайл*».

Атрибуты X и Y класса «Тайл» определяют соответственно номер столбца и строки ячейки объекта «*Ximage:Тайл*» в таблице. Вместе атрибуты (X,Y) задают координаты объекта «*Ximage:Тайл*» в контейнере «*tilelib*:*Карта*».

Описание атрибутов класса «Карта» дано в Таблице 7.

Таблица 7. Описание атрибутов класса «Карта»

Имя атрибута	**Описание**
scale	Текущий масштабный уровень отображения карты (далее масштаб). Атрибут применяется в алгоритме операции FindTile(). Атрибут инициализируется операцией scale() (см. [3]).
minX, maxX	Минимальное и максимальное значение номера столбца в таблице тайлов целевых данных для заданного масштаба. Атрибуты применяются для расчета размеров прямоугольника растра объекта «:*Карта*» в пикселях, а также в алгоритме преобразования координат тайлов.
minY, maxY	Минимальное и максимальное значение номера строки в таблице тайлов целевых данных для заданного масштаба. Атрибуты применяются для расчета размеров прямоугольника растра объекта «:*Карта*» в пикселях.
lat1, lon1	Географические координаты точки нижнего левого угла прямоугольника растра целевой карты. Атрибуты применяются для определения географических координат пикселей объекта «:*Карта*», а также для вычисления минимальных значений номеров строки и столбца таблицы тайлов спецификации MBTiles.
lat2, lon2	Географические координаты точки верхнего правого угла прямоугольника растра целевой карты.
minScale, maxScale	Минимальный и максимальный масштаб целевой карты. Атрибуты применяются для определения выхода за диапазон допустимых значений данных атрибута scale.
latCP, lonCP	Географические координаты центральной точки прямоугольника растра целевой карты. Значения атрибутов применяются для вычисления смещения СК объекта «*Карта:Растр*» относительно СК объекта «*Окно:Растр*» (см. [3]).
format	Тип формата растра целевого тайла. Атрибут применяется в конструкторе объекта «:*Конвертор*»

Описание операций класса «Карта» приводится в таблице 8.

Таблица 8. Описание операций класса «Карта»

Имя операции	**Описание**
GetMinX()	Возвращает значение атрибута minX для текущего масштабного уровня scale
GetMinY()	Возвращает значение атрибута minY для текущего масштабного уровня scale

4. Модель данных пакета «Драйвер»

Модель данных пакета «Драйвер» показана на рисунке 3 в виде диаграммы классов.

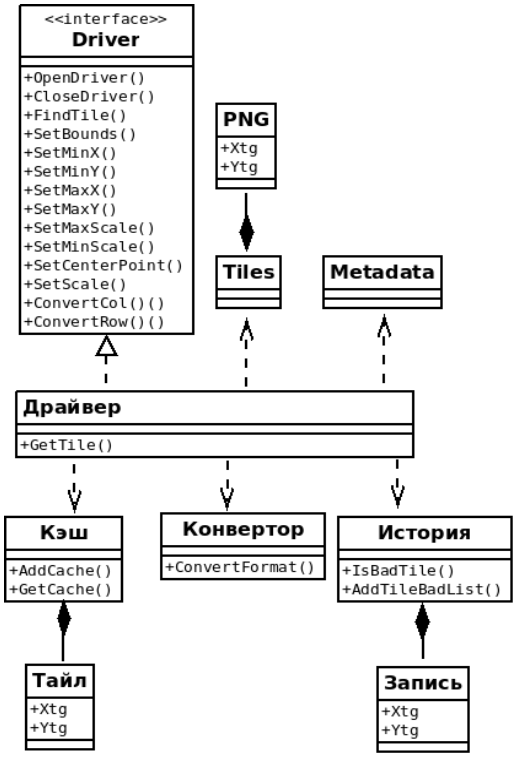

Рис. 3. Диаграмма классов пакета «Драйвер»

Интерфейсный класс «Driver» определяет набор операций (интерфейс), которые реализуются объектами класса «Драйвер». Указанная зависимость на диаграмме обозначена с помощью пиктограммы отношения реализации между соответствующими классами (пунктирная стрелка с незакрашенным треугольником на конце). В данном случае объекты класса «Драйвер» реализуют интерфейс, определяемый классом «Driver».

Описание операций интерфейса «Driver» приводится в таблице 9.

Таблица 9. Описание операций интерфейса «Driver»

Название операции	Описание
OpenDriver()	Создание объекта «*:Драйвер*».
CloseDriver()	Удаление объекта «*:Драйвер*».
FindTile()	Создание объекта «*Ximage:Тайл*» с координатами (X,Y) в контейнере «*tilelib:Карта*» или, если для этих координат объект сконструировать нельзя, - создание объекта «*UNDEF:Тайл*». Далее имя «UNDEF» будет применяться ко всем объектам с неопределенными данными, а ситуацию, когда создается такой объект - исключительной.
SetMinX(), SetMinY(), SetMaxX(), SetMaxY()	Инициализация атрибутов minX, maxX, minY и maxY объекта «*tilelib:Карта*».
SetMinScale(), SetMaxScale()	Инициализация атрибутов minScale и maxScale объекта «*tilelib:Карта*».
SetBounds()	Инициализация атрибутов lat1, lon1, lat2, lon2 объекта «*tilelib:Карта*».
SetCenterPoint()	Инициализация атрибутов latCP и lonCP объекта «*tilelib:Карта*».
SetScale()	Инициализация атрибутов scale объекта «*tilelib:Карта*».
ConvertCol()	Преобразование координаты X объекта «*Ximage:Тайл*» в координату Xtg целевого объекта «*:PNG*»
ConvertRow()	Преобразование координаты Y объекта «*Ximage:Тайл*» в координату Ytg целевого объекта «*:PNG*»

В настоящей работе класс «Драйвер» описывает объект «*MBTiles:Драйвер*», реализующий операции интерфейса «Driver» для целевых данных в формате MBTiles. У класса имеется закрытая операция GetTile(), которая создает объект «*:PNG*» с координатами (Xtg, Ytg) в контейнере «*:Tiles*» или «*UNDEF:PNG*» в случае исключительной ситуации.

Пиктограммы классов «Tiles» и «Metadata» на диаграмме абстрагируют таблицы tiles и metadata спецификации MBTiles.

Методы объекта «*MBTiles:Драйвер*» используют данные объектов «*:Tiles*» и «*:Metadata*» (см. Таблица 10), что на диаграмме показано с помощью пиктограмм отношения зависимости. В данных отношениях класс «Драйвер» зависит от классов «Tiles» и «Metadata».

Класс «PNG» описывает данные тайла в растровом формате PNG.

Класс «Tiles» является контейнером для класса «PNG», что на диаграмме показано с помощью пиктограммы ассоциации композитного агрегирования.

Класс «PNG» имеет два атрибута Xtg и Ytg, которые определяют координаты объекта «:*PNG*» в контейнере *«:Tiles*». С помощью этих атрибутов абстрагируются поля таблицы tiles спецификации MBTiles с именами tile_column и tile_row.

В таблице 10 описываются методы объекта «*MBTiles:Драйвер*», реализующие операции с БД, и имена таблиц и атрибутов схемы БД спецификации MBTiles, которые участвуют в SQL-запросах соответствующих методов.

Таблица 10. Целевые таблицы и поля SQL-запросов в методах объекта «*MBTiles:Драйвер*»

Метод	**Имя таблицы**	**Имя атрибута таблицы**
SetMinX(), SetMinY(), SetMaxX(), SetMaxY()	tiles	tile_column, tile_row. Для вычисления значений атрибутов применяются SQL – функции MIN() и MAX().
SetMinScale(), SetMaxScale()	metadata	name (параметры minzoom, maxzoom). Если строки с такими параметрами отсутствуют в целевой таблице, используется таблица tiles.
	tiles	zoom_level. Для вычисления значений параметров minzoom и maxzoom применяются SQL - функции MIN() и MAX().
SetBounds()	metadata	name (параметр bounds).
SetCenterPoint()	metadata	name (параметр center).
SetScale()	metadata	name (параметр center).
GetTile()	tiles	tile_data, zoom_level, tile_column tile_row

В реализации объекта «*MBTiles:Драйвер*» используются методы кэширования объектов «*Ximage:Тайл*» и предсказания исключительных ситуаций.

Применение кэширования в алгоритме метода FindTile() позволяет повысить производительность программы за счет повторного использования обработанных данных, а именно кэш объектов «*Ximage:Тайл*» позволяет при определенных условиях пропустить посылку сообщений GetTile() и ConvertFormat() в потоке управления метода FindTile().

Метод предсказания, так же как и кэширования, обеспечивает повышение производительности программы. Данный подход позволяет на основе накопленных данных в процессе выполнения метода GetTile() предсказать возникновение исключительной ситуации при создании объекта «*:PNG*» с координатами (Xtg, Ytg) и не отправлять сообщения GetCache() и GetTile().

Операции кэширования реализуются с помощью объектов класса «Кэш», а предсказания - с помощью объектов класса «История».

Импорт объектом «*MBTiles:Драйвер*» методов объектов «:*Кэш*» и «:*История*» на диаграмме показан с помощью пиктограмм отношения зависимости. В этих отношениях класс «Драйвер» зависит от классов «Кэш» и «История».

Класс «Кэш» определяет объекты, выполняющие кэширование объектов «*Ximage:Тайл*».

Данные объекта «:*Кэш*» на логическом уровне организованы как список объектов «*Ximage:Тайл*», то есть объект «:*Кэш*» является контейнером для объектов «*Ximage:Тайл*». На диаграмме это свойство объектов обозначено пиктограммой ассоциации композитного агрегирования.

Класс «Тайл» описывает объекты «*Ximage:Тайл*», созданные методом ConvertFormat() объекта «*PNG:Конвертор»* на основе данных объекта «:*PNG*».

Атрибуты Xtg и Ytg класса «Тайл» определяют координаты (Xtg,Ytg) объекта «*Ximage:Тайл*» в контейнере «:*Кэш*».

Классу «Кэш» принадлежат операции AddCache() и GetCache().

Операция AddCache() описывает функционал, который добавляет объект «*Ximage:Тайл*» в

контейнер «:*Кэш*», а операция GetCache() - сервис, который извлекает объект «*Ximage:Тайл*» из контейнера.

Класс «История» описывает объект-контейнер для объектов «:*Запись*».

То, что класс «История» является контейнером для класса «Запись», на диаграмме показано с помощью пиктограммы ассоциации композитного агрегирования.

Класс «Запись» определяет объекты «:*Запись*» контейнера «:*История*», с помощью которых регистрируются события возникновения исключительной ситуации при создании объекта «:*PNG*» с координатами (Xtg, Ytg).

Класс «Запись» имеет два атрибута Xtg и Ytg , которые вместе формируют координаты (Xtg, Ytg) объекта «:*Запись*» в контейнере «:*История*».

Классу «История» принадлежат операции IsBadTile() и AddTileBadList(). Операция IsBadTile() описывает функциональность поиска объекта «:*Запись*» по адресу (Xtg, Ytg) в контейнере «:*История*». С помощью операции AddTileBadList() абстрагируется метод-конструктор объекта «:*Запись*» с адресом (Xtg, Ytg) в контейнере «:*История*».

Для преобразования растровых форматов целевых тайлов в формат XImage объекта «*Ximage:Тайл*» объект «*:Драйвер*» использует объекты-конверторы форматов. Для каждого типа растрового формата целевого тайла, который определяется данными поля format объекта «*tilelib:Карта*», создается соответствующий объект «*:Конвертор».* Такой подход обеспечивает инвариантность объекта «:*Драйвер*» к растровым форматам целевых тайлов.

Далее будет рассматриваться объект «*PNG:Конвертор»,* выполняющий преобразование данных из растрового формата PNG в формат XImage.

Объекты «*:Конвертор»* описываются классом «Конвертор».

В классе «Конвертор» преобразование данных целевого тайла выполняет операция ConvertFormat().

Объект «*MBTiles:Драйвер*» класса «Драйвер» для создания объекта «*Ximage:Тайл*» использует метод ConvertFormat() объекта «*PNG:Конвертор»* класса «Конвертор». На диаграмме импорт метода ConvertFormat() обозначен с помощью пиктограммы отношения зависимости между классами «Драйвер» и «Конвертор». В этом отношении Класс «Драйвер» зависит от класса «Конвертор».

Входными данными метода ConvertFormat() объекта «*PNG:Конвертор»* являются данные объекта «:*PNG*» с координатами (Xtg, Ytg) в контейнере «*:Tiles*». Выходными данными является объект «*XImage:Тайл*» с координатами (X,Y) в контейнере «*tilelib:Карта*». В методе ConvertFormat() для обработки данных в формате PNG используются методы объектов классов пакета «libpng».

5. Алгоритм преобразования координат в методах ConvertCol() и ConvertRow()

В данном разделе рассматривается алгоритм, в котором координаты (X,Y) объекта «*XImage:Тайл*» в контейнере «*tilelib:Карта*» преобразуются в координаты (Xtg, Ytg) объекта «:*PNG*» в контейнере «*:Tiles*». Данный алгоритм используется в методах ConvertCol() и ConvertRow() объекта «*MBTiles:Драйвер*».

В алгоритме предполагается, что объекты «*tilelib:Карта*» и «*:Tiles*» имеют системы координат, в которых тайл рассматривается в виде точки с координатами (x,y), где координате «x» соответствует номер столбца, а координате «y» - номер строки таблицы тайлов.

В алгоритме приняты следующие обозначения для координат объектов «*XImage:Тайл*» и «:*PNG*» в СК контейнеров «*tilelib:Карта*» и «*:Tiles*». В СК контейнера «*tilelib:Карта*» координаты объекта «*XImage:Тайл*» обозначаются (X,Y), а координаты соответствующего ему объекта «*:PNG*» - (Xt, Yt); в СК контейнера «*:Tiles*» координаты объекта «*:PNG*» обозначаются (Xtg, Ytg).

В соответствии с моделью данных библиотеки «tilelib» в СК контейнера «*tilelib:Карта*» положительная ось абсцисс направлена вправо от начала координат, а положительная ось ординат — вниз. Значения координат (X,Y) объекта «*XImage:Тайл*» начинаются с нуля. Для вычисления координат (X,Y) применяется следующая формула:

$$X = Xt - GetMinX(scale) \qquad (1)$$
$$Y = Yt - GetMinY(scale)$$

, где scale – это текущий уровень масштабирования, GetMinX(), GetMinY() - функции, которые возвращают минимальные значения координат (Xtg, Ytg) объекта «*:PNG*» в контейнере «*:Tiles*». Данные функции реализуются методами GetMinX(), GetMinY() объекта «*tilelib:Карта*».

В СК контейнера «*:Tiles*» в соответствии со спецификацией TMSS положительная ось абсцисс направлена вправо от начала координат, а положительная ось ординат — вверх. Начальные значения координат (Xtg, Ytg) объекта «:*PNG*» в СК контейнера «*:Tiles*» определяются функциями (GetMinX(scale),GetMinY(scale)).

Задача преобразования координат состоит в том, чтобы для объекта «*XImage:Тайл*» с координатами (X,Y) в СК контейнера «*tilelib:Карта*»

определить координаты (Xtg, Ytg) соответствующего ему объекта «:*PNG*» в СК контейнера «*:Tiles*».

Алгоритм преобразования координат следующий.

Вычислим по формуле (2) координаты (Xt, Yt) объекта «*:PNG*» в СК контейнера «*tilelib:Карта*», используя координаты (X, Y) объекта «*XImage:Тайл*» для текущего уровня масштабирования scale.

$$Xt = X + GetMinX(scale) \quad (2)$$
$$Yt = Y + GetMinY(scale)$$

Чтобы определить координаты (Xtg, Ytg) объекта «*:PNG*» в СК контейнера «*:Tiles*», вычислим максимальное значение m координаты Ytg объекта «*:PNG*» для уровня масштабирования scale по следующей формуле [8].

$$m = 2^{scale} - 1 \quad (3)$$

После этого с помощью формулы (4) рассчитаем координаты (Xtg, Ytg) объекта «:*PNG*» в СК контейнера «*:Tiles*».

$$Xtg = Xt \quad (4)$$
$$Ytg = m - Yt$$

6. Алгоритм преобразования данных в методе FindTile()

В данном разделе рассматривается алгоритм реализации метода FindTile() объекта «*MBTiles:Драйвер*».

Назначение метода FindTile() состоит в том, чтобы создать объект «*Ximage:Тайл*» с координатами (X,Y).

Для этого надо для объекта «*Ximage:Тайл*» с координатами (X,Y) найти в контейнере «*:Tiles*» соответствующий объект «*:PNG*» с координатами (Xtg, Ytg) , а затем, используя данные объекта «*:PNG*», с помощью метода ConvertFormat() объекта «*PNG:Конвертор*» создать объект «*Ximage:Тайл*».

Ниже приводится описание алгоритма выполнения метода FindTile().

Чтобы создать объект «*:PNG*», вычислим координаты (Xtg, Ytg) объекта «*Ximage:Тайл*» в СК контейнера «*:Tiles*», которые являются входными данными для метода IsBadTile() объекта «:*История*», GetCache() объекта «:*Кэш*» и GetTile() объекта «*MBTiles:Драйвер*».

Для этого преобразуем координаты (X,Y) объекта «*Ximage:Тайл*» с помощью методов ConvertCol() и ConvertRow() объекта «*MBTiles:Драйвер*». В результате получим координаты (Xtg, Ytg).

Затем создадим объект «*:PNG*». Данные для создания объекта могут отсутствовать в контейнере «*:Tiles*». Чтобы проверить наличие данных для объекта сначала выполним метод IsBadTile() объекта «:*История*». Если для координат (Xtg, Ytg) существует объект «:*Запись*» с идентификатором (Xtg, Ytg) в контейнере «:*История*», тогда завершаем поток управления операции FindTile() и формируем объект «*UNDEF:Тайл*». Если объект «:*Запись*» отсутствует, проверяем наличие данных в контейнере «:*Кэш*». Для этого выполняем метод GetCache(), который извлекает объект «*Ximage:Тайл*» с координатами (Xtg, Ytg) из контейнера, или, если объект отсутствует, возвращает объект «*UNDEF:Тайл*». Если получен объект «*UNDEF:Тайл*», выполняем поиск объекта «*:PNG*» в контейнере «*:Tiles*» с помощью метода GetTile() объекта «*MBTiles:Драйвер*». Метод создает объект «*:PNG*», если данные для объекта с координатами (Xtg, Ytg) имеются в контейнере «*:Tiles*», или «*UNDEF:PNG*», если они отсутствуют.

Если получен объект «*UNDEF:PNG*» добавим объект «:*Запись*» с координатами (Xtg, Ytg) в контейнер «:*История*» с помощью метода AddTileBadList() и создадим объект «*UNDEF:Тайл*».

Если объект «*:PNG*» создан, то сконструируем объект «*Ximage:Тайл*» и добавим его в контейнер «:*Кэш*» с координатами (Xtg, Ytg). Для этого с помощью метода ConvertFormat() объекта «*PNG:Конвертор*» преобразуем объект «*:PNG*» в объект «*Ximage:Тайл*».

После этого вызовем метод AddCache() и разместим объект «*Ximage:Тайл*» в контейнере «:*Кэш*».

Заключение

В данной статье был рассмотрен метод построения инвариантной библиотеки «tilelib», которая предоставляет набор функций для отображения тайловой карты на устройствах вывода графической информации.

При разработке метода была учтена специфика функционирования программных изделий, созданных в ФГУ ФНЦ НИИСИ РАН.

В результате была реализована библиотека «tilelib» со следующими характеристиками:

1. Библиотека обеспечивает инвариантность программ, импортирующих ее функции, к целевым электронным картам с тайловой моделью данных.

2. Внутренний формат растра тайловой карты библиотеки соответствует формату растра графической библиотеки ГБ, разработанной в ФГУ ФНЦ НИИСИ РАН.

3. В библиотеке используется открытый программный код, а именно импортируются функции библиотек «libpng» и «SQLite».

При описании метода были раскрыты следующие темы:

1. Спецификация MBTiles.

2. Архитектура инвариантной библиотеки «tilelib».

Моделирование статических и динамических аспектов архитектуры библиотеки было выполнено средствами языка UML.

В результате проектирования архитектуры библиотеки были разработаны:

1. Модели данных пакетов «tilelib» и «Драйвер». По сравнению с [3] в описатели классов были добавлены новые атрибуты и операции, обеспечивающие инвариантность библиотеки.

2. Алгоритм преобразования координат в методах ConvertCol() и ConvertRow().

3. Алгоритм реализации метода FindTile().

Для тестирования библиотеки «tilelib» были реализованы контрольная задача и программа драйвер.

В качестве целевых данных в контрольной задаче использовались данные в формате MBTiles.

Преобразование целевых данных в формате MBTiles в формат данных библиотеки выполнялось с помощью программы драйвера.

Интерфейс программы драйвера был реализован в соответствии с моделью данных пакета «Драйвер» архитектуры библиотеки.

Контрольная задача запускалась на ЭВМ "Багет-47-51" под управлением операционной системы реального времени.

Успешное выполнение КЗ подтвердило применимость предложенного метода построения инвариантной библиотеки в программной среде, разработанной в ФГУ ФНЦ НИИСИ РАН.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН (проведение фундаментальных научных исследований 47 ГП) по теме № 0065-2019-0002 «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (рег. № АААА-А19-119012290074-2)

A Method for Building a Raster Map Display Library That Is Invariant to Target Geospatial Data Formats

P.V. Egorov

Abstract. A method of building a “tilelib” library that is invariant to the target spatial data formats. The “tilelib” library is intended for displaying a digital raster map using graphic information output devices. As an example of target data, data conforming to the MBTiles specification is used.

Keywords: raster electronic map, tile map, tile, real-time.

Литература

1. Raster drivers. https://gdal.org/drivers/raster/index.html. (Дата обращения 25.05.2021).

2. GDAL. https://gdal.org/ (Дата обращения 25.05.2021).

3. Егоров П.В. Построение изображения растровой электронной карты с использованием программных средств для операционной системы реального времени семейства «Багет». «Труды НИИСИ», Т. 10 (2020), № 5-6, 127–138.

4. MBTiles 1.3. https://github.com/mapbox/mbtiles-spec/blob/master/1.3/spec.md (Дата обращения 25.05.2021).

5. Tile Map Service Specification. https://wiki.osgeo.org/wiki/Tile_Map_Service_Specification (Дата обращения 25.05.2021).

6. Гради Буч, Джеймс Рамбо, Ивар Якобсон «Язык UML. Руководство пользователя». Москва «Издательство ДМК Пресс» 2006 г.

7. М. Фаулер, К. Скотт «UML в кратком изложении». Москва «Мир» 1999 г.

8. Slippy map tilenames. https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames (Дата обращения 25.05.2021).

Программные инструменты энергоэффективного планирования суперкомпьютерных заданий

Е.А. Киселёв[1], А.В. Баранов[2], О.С. Аладышев[3], П.Н. Телегин[4], А.В. Яровой[5]

[1]Межведомственный суперкомпьютерный центр РАН, Москва, Россия, kiselev@jscc.ru;

[2]Межведомственный суперкомпьютерный центр РАН, Москва, Россия, antbar@mail.ru;

[3]Межведомственный суперкомпьютерный центр РАН, Москва, Россия, aladyshev@jscc.ru;

[4]Межведомственный суперкомпьютерный центр РАН, Москва, Россия, telegin@jscc.ru;

[5]Московский Технический Университет Связи и Информатики, Москва, Россия, iarovoi-av@yandex.ru

Аннотация. Статья посвящена описанию разработанных программных средств сбора и анализа данных об энергопотреблении вычислительных ресурсов. Авторами предложен алгоритм энергоэффективного планирования суперкомпьютерных заданий, позволяющий выбирать оптимальную вычислительную систему с учетом энергозатрат на вычисления и допустимого увеличения времени выполнения. Представлены результаты экспериментов с тестовым набором приложений и результатами запуска реальных заданий пользователей МСЦ РАН.

Ключевые слова: энергоэффективность, планирования заданий, энергопотребление, суперкомпьютер.

1. Введение

Для любого центра обработки данных и суперкомпьютерного центра (СКЦ) весьма критична ситуация длительной работы вычислительных ресурсов в режиме максимальной вычислительной нагрузки. В такие периоды энергопотребление суперкомпьютера резко возрастает, создавая пиковые нагрузки на подсистемы электропитания и охлаждения. Часто, особенно в летний период, подобные пиковые нагрузки приводят к отключениям части вычислительных ресурсов. В таких условиях актуальной становится задача учёта, контроля и планирования энергопотребления одновременно работающих вычислительных систем при запуске заданий пользователей СКЦ, т.е. задача осуществления энергоэффективного планирования суперкомпьютерных заданий. Важную роль в таком планировании играют программные инструменты, позволяющие осуществлять сбор, накопление и анализ данных об энергопотреблении различных суперкомпьютерных приложений, запускаемых пользователями при выполнении вычислительных заданий.

В рамках продолжения исследований [1,2], посвященных разработке и реализации алгоритмов энергоэффективного планирования вычислительных ресурсов СКЦ, авторами разработан программный комплекс, включающий в себя средства сбора и анализа данных об энергопотреблении вычислительных ресурсов во время выполнения программ, а также программный модуль энергоэффективного планирования вычислительных ресурсов СКЦ. В работе приведены результаты оценки эффективности работы алгоритма для заданий из тестового пакета NAS Parallel Benchmarks и заданий пользователей СКЦ.

2. Средства сбора и анализа данных об энергопотреблении вычислительных ресурсов

Проведенный авторами анализ [2] известных методов и средств измерения энергопотребления вычислительных систем позволил разработать программные средства Messtat и Emonitor.

Messtat предназначено для сбора данных об энергопотреблении микропроцессора и загрузке вычислительных ядер из регистров MSR, счетчиков производительности perf_event и интерфейса powercap. Программное средство каждую секунду производит сбор данных об энергопотреблении микропроцессоров (в Дж) и загрузке вычислительных ядер (в %). Данные заносятся в текстовый файл, который создается в момент запуска программы на вычислительном узле. Каждая строка файла соответствует одной секунде выполнения программы и содержит значения

энергопотребления для каждого сокета процессора, процент загруженности всех доступных процессорных ядер (см. Листинг 1).

Листинг 1

Пример содержимого файла Messtat

1:package-1: 19.990062J dram: 1.760097J

package-0: 27.849538J dram: 2.376258J

1.57	0.00	0.99	0.00	0.00	2.78
0.00	0.00	2.02	0.00	0.00	4.65
0.00	0.00	0.00	0.00	0.00	0.00
2.33	4.55	0.00	0.00	0.00	0.00
0.99	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	4.88	4.88	4.88
4.88	0.00	6.98	4.76	4.65	4.76
6.82	0.00	4.55	4.65	4.65	4.65
6.82	4.65	1.98	0.99	2.33	4.55
4.55	4.55	2.33	2.38	8.70	6.67
2.33	4.65	4.55	4.55	4.55	

2: package-1: 86.068322J dram: 5.341490J

package-0: 96.799068J dram: 6.028322J

35.45	76.77	71.72	21.43	72.28	72.45
72.00	72.28	72.73	65.00	65.35	75.26
65.00	71.72	64.65	64.65	65.00	72.00
67.35	66.67	65.00	64.65	64.65	64.29
64.65	72.00	64.65	64.65	72.28	72.28
71.29	64.00	64.36	2.04	2.02	66.33
1.02	1.02	2.02	2.02	2.02	2.02
1.02	0.00	2.02	1.02	2.02	2.02
1.02	4.04	0.00	0.00	1.02	1.02
1.02	2.02	2.02	2.02	1.02	1.02
2.02	2.02	2.00	1.02	1.01	

Для анализа и интерпретации данных из файлов Messtat, используется программное средство Emonitor. Данное программное средство позволяет оценить максимальное, минимальное, среднее и суммарное значения энергопотребления как для одной, так и для нескольких параллельных программ пользователя.

Данные об энергопотреблении могут быть представлены как в текстовом, так и в графическом виде с отображением диаграмм (рис. 1) и графиков (рис. 2). С использованием графического интерфейса данные об энергопотреблении компонентов параллельной программы на каждом вычислительном узле могут быть соотнесены с числом используемых вычислительных ядер и процентом их загрузки. Все данные об энергопотреблении параллельных программ загружаются в базу данных статистики и могут использоваться для планирования запуска заданий.

Рис.1. Пример отображения круговой диаграммы с данными об энергопотреблении программ

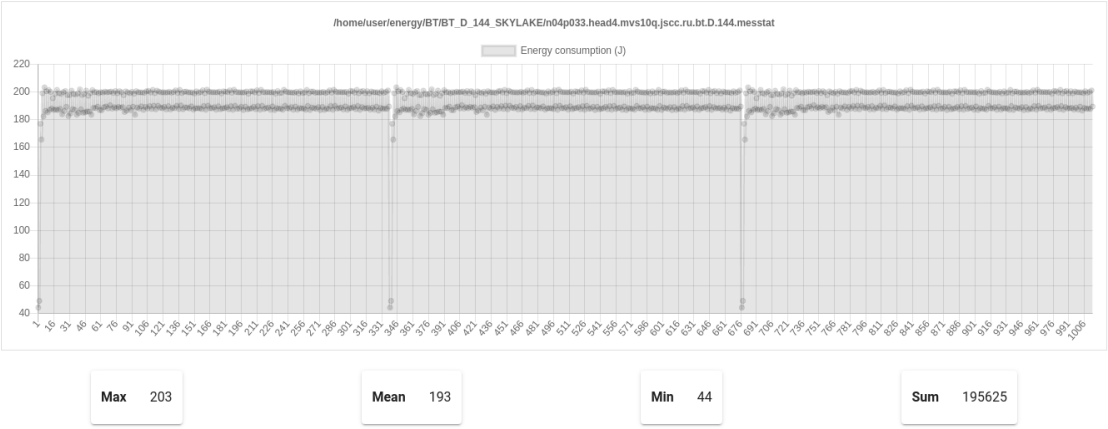

Рис. 2. Пример отображения графика посекундного изменения энергопотребления программы

3. Алгоритм энергоэффективного планирования вычислительных ресурсов СКЦ

С целью решения задачи энергоэффективного планирования вычислительных ресурсов СКЦ авторами разработан алгоритм, позволяющий выбирать оптимальную для запуска вычислительную систему с учетом энергозатрат параллельного приложения (C) и допустимого увеличения времени выполнения параллельной программы (K). В основе предложенного алгоритма используются идеи расширения EAS [4], адаптированные к условиям применения в СКЦ.

В общем виде алгоритм может быть представлен следующей последовательностью шагов.

Шаг 1. Сформировать список вычислительных систем (Systems) для запуска параллельной программы.

Шаг 2. Для всех вычислительных систем из списка Systems определить значения C на предыдущих запусках параллельной программы. Если в списке Systems присутствует вычислительная система, на которой ранее не производился запуск параллельной программы, установить значение C=0.

Шаг 3. Для всех вычислительных систем из списка Systems определить значения T по предыдущим запускам параллельной программы. Если в списке Systems присутствует вычислительная система, на которой ранее не производился запуск параллельной программы, установить значение T=0.

Шаг 4. Выбрать из списка Systems вычислительную систему, для которой достигается наименьшее значение C при заданном пороговом значении K. Завершить работу алгоритма.

Следует заметить, что эффективность работы алгоритма зависит от наличия информации о предыдущих запусках параллельной программы. Например, если такая информация отсутствует, то алгоритм выделяет вычислительные ресурсы СКЦ из списка ранее не использованных. Если значение K не задано или равно 0, то параллельная программа запускается на вычислительной системе с наименьшим значением C. Вычисление значения энергопотребления и определение времени выполнения параллельной программы осуществляется в автоматическом режиме в соответствии с методикой [2]. Полученные значения сохраняются для последующего использования в алгоритме.

Представленный алгоритм был реализован на языке программирования C++ и интегрирован в состав командного интерфейса Системы управления прохождением параллельных заданий (СУППЗ) [5], применяемой в Межведомственном суперкомпьютерном центре РАН (МСЦ РАН). В СУППЗ для запуска параллельных программ используется в том числе команда mpirun, при вызове которой пользователь указывает количество требуемых заданию процессоров, максимальное время аренды вычислительных ресурсов системы, тип используемого вычислительного ресурса и имя исполняемого файла с входными параметрами.

Для наших целей команда mpirun была модифицирована так, чтобы при ее вызове вычислялась hash-сумма исполняемого файла параллельной программы. Вычисленное значение hash-суммы сохраняется в отдельном файле и используется как уникальный идентификатор программы. Значение hash-суммы, а также используемые для запуска программы параметры mpirun сохраняются в базу данных Emonitor. Далее инициируется запуск алгоритма выбора вычислительной системы.

Если при запуске параллельной программы

пользователь задал тип используемого вычислительного ресурса, то результат работы алгоритма будет носить уведомительный характер. Пользователю будет предложена вычислительная система с наименьшим энергопотреблением. Если пользователь не задавал тип используемого вычислительного ресурса, то в параллельная программа будет автоматически поставлена в очередь вычислительной системы с наименьшим энергопотреблением для данного приложения. Значение параметра K, характеризующего допустимое превышение времени работы параллельной программы, может быть задано администратором или вычисляться автоматически перед запуском алгоритма. Так, если ранее производился запуск параллельной программы и время ее выполнения (T) не превышало заказанное время аренды вычислительных ресурсов (T_{max}), то значения K вычисляется по формуле:

$$K = \frac{T_{max}}{T}$$

4. Экспериментальное исследование разработанного алгоритма

Экспериментальное исследование разработанного алгоритма осуществлялось в два этапа.

На первом этапе проводилась серия экспериментов с использованием тестов NAS Parallel Benchmarks (NPB) BT, EP, IS, LU и SP класса сложности D. Выбор указанного класса сложности был обусловлен максимальной загрузкой узлов суперкомпьютера, производительностью и характером использования вычислительных ресурсов в процессе их выполнения.

На втором этапе оценивались результаты запуска 24 заданий пользователей МСЦ РАН, сохраненные в базе данных статистики.

Таблица 1. Параметры запуска тестов NPB и приложений пользователей МСЦ РАН

Название приложения	Количество выделенных ВУ			
	Broadwell	Cascade lake	KNL	Skylake
Тесты Nas Parallel Benchmarks				
BT	5	3	2	4
EP	5	3	2	4
IS	8	6	4	8
LU	8	6	4	8
SP	8	6	4	8
Приложения пользователей МСЦ РАН				
decomposePar	1	1	1	1
dxa.py	1	1	1	1
eQ.gms	1	1	1	1
eQ.p	1	1	1	1
flowmod2021mpi	6	8	5	3
g09e01avx.sh	1	1	1	1
g16.sh	1	1	1	1
gmx_mpi	6	4	4	4
laptest	2	1	1	1
lim-ballistics2d	2	2	1	2
lmp_mpi_10q	6	3	4	6
nesvetay2020	7	5	4	4
nutcy	1	1	1	1
p4	1	2	1	1
ph.x	3	4	4	4
pp.x	5	1	3	6
pw.x	6	3	3	4
reconstructPar	1	1	1	1
RectRotat	9	7	4	8
SC4	6	7	3	7
strip-sol	1	1	1	1
vasp_gam	4	3	2	4
vasp_ncl	7	6	3	4
vasp_std	5	5	2	6

В качестве экспериментальной платформы были использованы суперкомпьютеры МВС-10П МП2 (раздел KNL), МВС-10П ОП (раздел Broadwell), МВС-10П ОП (раздел Skylake) и МВС-10П ОП (раздел Cascade lake), установленные в МСЦ РАН [5]. Перечисленные

вычислительные системы обладают различной производительностью и энергопотреблением. В таблице 1 приведено количество ВУ каждой вычислительной системы, выделенных СУППЗ для выполнения тестовых заданий и приложений пользователей.

Оценка эффективности работы алгоритма производилась по двум основаниям: энергопотреблению и времени выполнения приложения.

На рисунках 3 и 4 представлены результаты сравнения энергопотребления (W) и времени выполнения приложений (T) из таблицы 1 при разных режимах работы алгоритма. Alg(0) соответствует режиму работы со значением параметра K, равным 0, а Alg(85) – режиму работы со значением параметра K, равным 85.

Таблица 2. Влияние снижения энергопотребления на время выполнения приложений

Название приложения	Снижение W, %	Увеличение T, %
eQ.gms	3,5	30
flowmod2021mpi	5,9	26
laptest	14	48
nesvetay2020	30	41
lim-ballistics2d	39	50
RectRotat	49	48,7
vasp_gam	42	64,8
vasp_ncl	46,6	58,6
vasp_std	20	27,1

Анализ энергопотребления (рис.5) и времени выполнения (рис.6) заданий пользователей МСЦ РАН при разных режимах работы алгоритма планирования позволяет отметить следующее. Для подавляющего большинства представленных приложений (15 из 24) запуск на более энергоэффективных вычислительных ресурсах не приводил к увеличению времени выполнения. Для 9 приложений, представленных в таблице 2, сокращение энергопотребления от 3,5% до 49% приводило к увеличению времени выполнения от 26% до 65% и было не всегда целесообразным. Данная особенность связана с тем, что указанные приложения ориентированы на конкретную аппаратную платформу и их запуск на альтернативных вычислительных ресурсах с меньшей производительностью приводил к значительному увеличению времени выполнения.

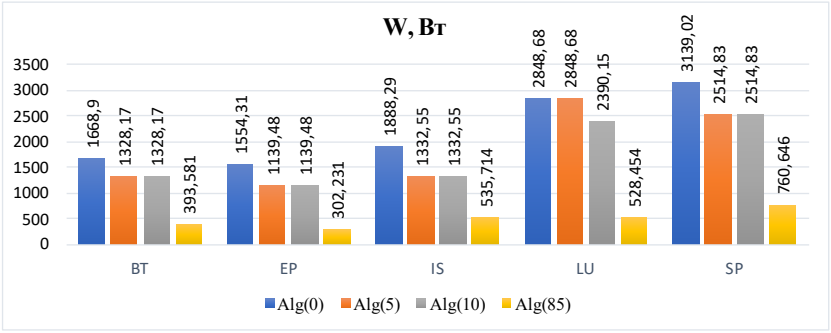

Рис. 3. Зависимость энергопотребления от параметров работы алгоритма при выполнении тестов NPB

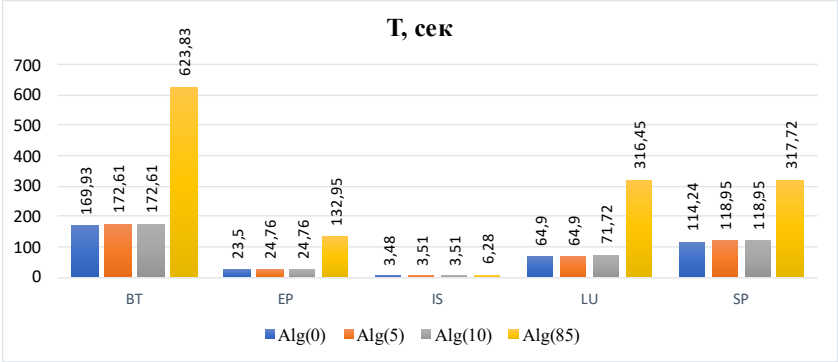

Рис. 4. Зависимость времени выполнения тестов NPB от параметров работы алгоритма

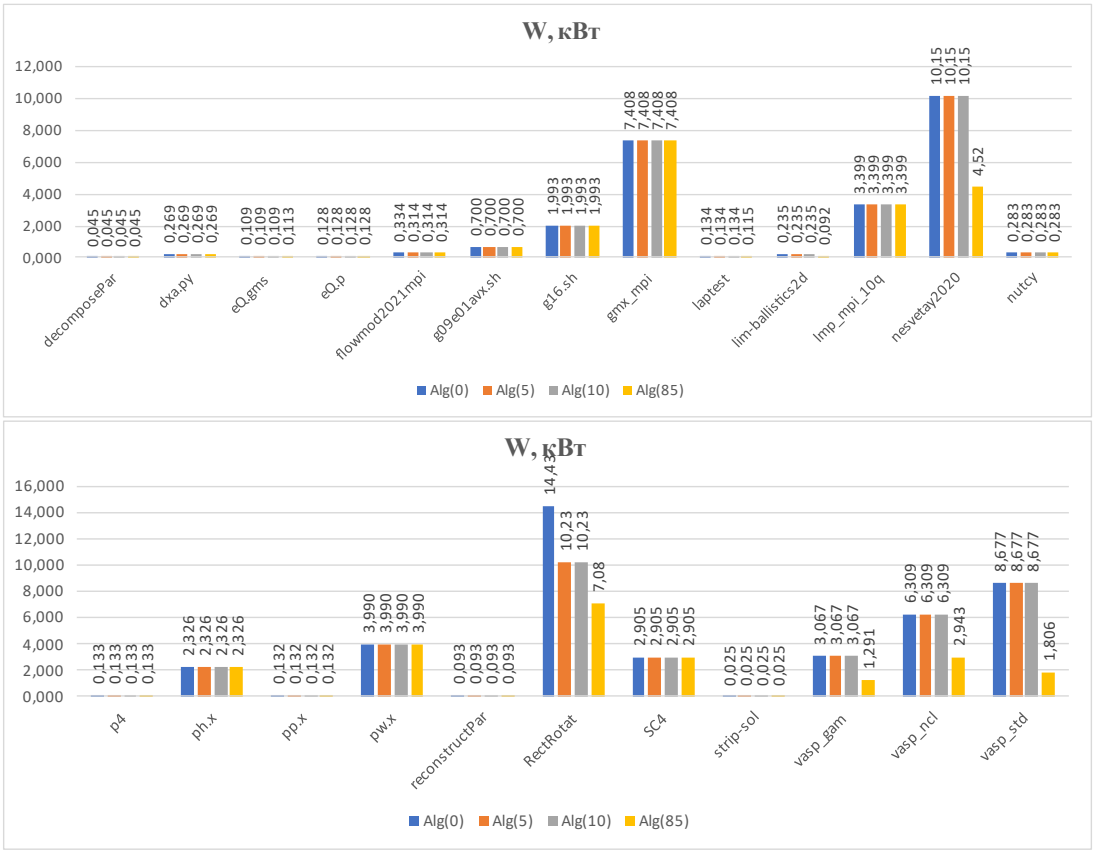

Рис. 5. Зависимость энергопотребления от параметров работы алгоритма при выполнении приложений пользователей

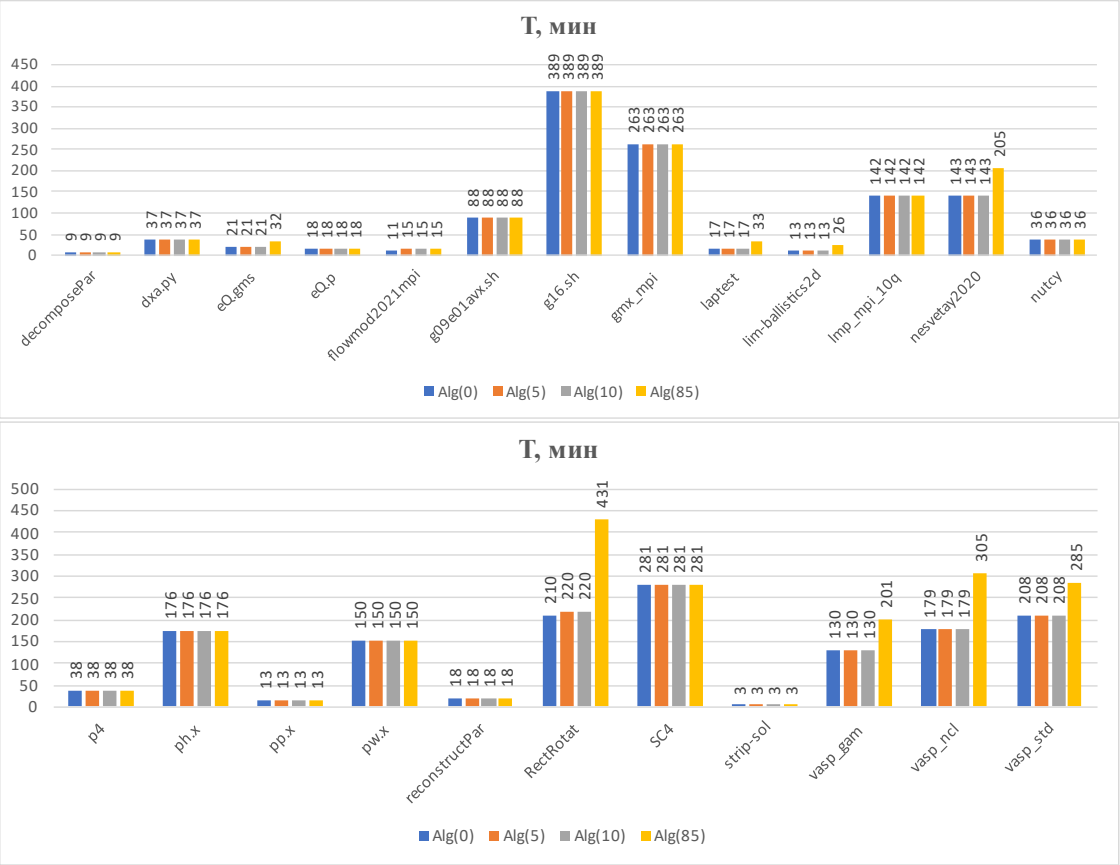

Рис. 6. Зависимость времени выполнении приложений пользователей от параметров работы алгоритма

5. Заключение

Результаты проведенного исследования позволяют сделать вывод о целесообразности применения разработанных программных средств и алгоритма энергоэффективного планирования вычислительных ресурсов в составе таких систем обработки заданий, как СУППЗ.

Целесообразно заметить, что для различных классов задач эффективность представленного алгоритма может отличаться. В большинстве случаев достигается сокращение энергопотребления при сохранении или незначительном (не более 10%) увеличении времени выполнения приложений. В то же время, для приложений, ориентированных на конкретную аппаратную архитектуру, снижение энергопотребления приводит к значительном увеличению времени выполнения (от 26% до 65%).

Исследования проведены при поддержке РФФИ (грант № 19-07-01072).

Software of Energy-Efficient Scheduling of Supercomputer Tasks

E. Kiselev, A. Baranov, O. Aladyshev, P. Telegin, A. Yarovoy

Abstract. The article discusses software tools designed to perform energy-efficient scheduling of user tasks on the resources of concurrently operating computing systems of supercomputer centers. The authors of the article presented an algorithm for energy-efficient scheduling of supercomputer tasks which makes it possible to choose a computing system, taking into account the energy consumption of parallel applications and an allowable increase in the execution time of a parallel program. In the course of experiments with a test set of tasks and the results of launching

real user tasks of the JSCC of RAS, the dependence between the energy consumption of launched applications and their execution time was derived.

Keywords: energy-efficient scheduling, power consumption, supercomputers.

Литература

1. E. Kiselev, V. Kiselev, B. Shabanov, O. Aladyshev, A. Baranov. The Energy Efficiency Evaluating Method Determining Energy Consumption of the Parallel Program According to its Profile. "Lobachevskii Journal of Mathematics", V. 41 (2020), N. 12, 2542-2551, DOI: 10.1134/S1995080220120161.

2. Киселёв Е.А., Киселев В.И., Баранов А.В., Аладышев О.С. Способ оценки энергопотребления заданий в суперкомпьютерной системе коллективного пользования. Труды научно-исследовательского института системных исследований Российской академии наук. 2020. Т. 10. № 5-6. С. 21-26, DOI: 10.25682/NIISI.2020.5_6.0003.

3. Linux Scheduler. Energy Aware Scheduling // https://www.kernel.org/doc/html/latest/scheduler/sched-energy.html.

4. Savin G.I., Shabanov B.M., Telegin P.N., Baranov A.V. Joint Supercomputer Center of the Russian Academy of Sciences: Present and Future. Lobachevskii Journal of Mathematics, 40 (11), pp. 1853-1862. DOI: 10.1134/S1995080219110271.

5. Вычислительные ресурсы МСЦ РАН. https://www.jscc.ru/resources/hpc.

Вопросы обеспечения кибербезопасности при разработке и использовании АСУ ТП

А.И. Грюнталь[1], С.Е. Базаева[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, grntl@niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, bazaeva@niisi.msk.ru

Аннотация. Статья содержит обзор основных требований и методов, применяемых для обеспечения кибербезопасности технологических процессов, управляемых с помощью АСУ ТП. На примере ПЛК, функционирующего в составе аппаратно-программной платформы «Багет» разработки ФГУП ФНЦ НИИСИ РАН, приводится оценка степени соответствия требованиям, выдвигаемым стандартом ГОСТ Р МЭК 62443 к производителям и поставщикам компонентов АСУ ТП.

Ключевые слова: АСУ ТП, кибербезопасность, ПЛК, платформа «Багет»

1. Введение

Задача обеспечения безопасности на промышленных предприятиях при использовании автоматизированных систем управления технологическими процессами (АСУ ТП) в настоящее время является одной из важнейших тем в сфере вычислительных технологий, поскольку цифровизация управления производством затрагивает физические объекты, что многократно увеличивает ущерб, наносимый кибератаками злоумышленников или обусловленный ошибками проектирования, разработки и эксплуатации системы. Значимость проблемы насколько велика, что в промышленно развитых странах она вышла на государственный уровень и регулируется законодательно.

Современные промышленные технологии и IT-системы позволяют автоматизировать объект практически любой сложности, вплоть до объектов критически важной инфраструктуры в атомной энергетике, химической, оборонной промышленности, связи и прочих областях, где потенциальный ущерб может оказаться катастрофическим. Поэтому проблема обеспечения безопасности АСУ ТП состоит в том, чтобы обеспечить непрерывность и целостность технологического процесса, а также его безопасность для жизни и здоровья людей и окружающей среды.

В отношении таких систем используют термин «кибербезопасность» в отличие от привычных терминов «информационная безопасность» и «функциональная безопасность», применяемых по отдельности для информационных систем и промышленных систем, использующих средства автоматики. Кибербезопасность АСУ ТП включает решение целого комплекса задач – и защиту периметра, и обеспечение бесперебойного функционирования оборудования, и отражение сетевых атак, и безопасную разработку программного обеспечения, и сохранение конфиденциальности информации.

Актуальность задач обеспечения кибербезопасности АСУ ТП обусловлена следующими факторами:

– при создании АСУ ТП используются универсальные технологии и протоколы, информация о которых доступна для злоумышленников;

– технологическую сеть АСУ ТП часто невозможно изолировать от IT-инфраструктуры компании, что облегчает взлом и проникновение;

– при проектировании и разработке сложной системы невозможно избежать ошибок и просчетов;

потенциальный ущерб (как в случае целенаправленной атаки на АСУ ТП, так и в случае аварийных ситуаций, обусловленных ошибками и неисправностью системы) может оказаться неприемлемым.

2. Архитектура и принципы функционирования АСУ ТП

Архитектура АСУ ТП имеет несколько уровней (рис 1). В состав системы входят:

– подсистема электропитания;

– «полевое» оборудование (датчики и исполнительные механизмы);

– программируемые логические контроллеры, которые управляют оборудованием и выполняют ввод/вывод;

– сетевое оборудование, серверы, компоненты пользовательского интерфейса (ЭВМ оператора/инженера) для управления функционированием автоматизированной системы.

Рис. 1. Структура АСУ предприятия

В случае масштабного производства АСУ отдельного технологического процесса может быть подключена к системе управления более высокого уровня – так называемой «цеховой» системе MES (manufacturing execution system), с помощью которой осуществляется оперативное управление производственными процессами.

На предприятиях, где задействовано множество технологических процессов, потребуются также системы ERP (Enterprise Resource Planning) для управления подразделениями и ресурсами и системы стратегического управления бизнесом класса BI (Business Intelligence).

Поскольку АСУ ТП располагается в основании управленческой структуры, данный элемент оказывается одним из главных факторов производственной деятельности, так как является одновременно объектом управления и важным источником информации для всей системы. Следовательно, безопасность функционирования АСУ ТП и надежность ее защиты от атак злоумышленников является залогом успешности производства в целом.

Современные АСУ ТП представляют собой распределенную систему управления и контроля физических объектов, включенных в отдельный технологический процесс.

Логика функционирования АСУ ТП состоит в том, чтобы с помощью «полевых» датчиков, подключенных к производственному оборудованию, передавать информацию о его состоянии и в случае необходимости оказывать управляющее воздействие, используя исполнительные механизмы. Данные с «полевого» уровня поступают на программируемые логические контроллеры, которые управляют оборудованием и осуществляют ввод-вывод и первичную обработку. Это уровень управления. На диспетчерском (верхнем) уровне АСУ ТП обычно находится SCADA-система (система диспетчерского управления и сбора данных) с человеко-машинным интерфейсом, работу которой контролирует оператор (в штатном режиме) или инженер (в режиме настройки или технологическом режиме).

Все перечисленные элементы авто-матизированной системы соединены между собой каналами передачи информации: сетью Ethernet, системой приема/передачи радио-сигналов или иными способами в зависимости от специфики области применения. В неко-торых случаях полевое оборудование может располагаться на расстоянии десятков километров от центра управления.

Программируемые логические контроллеры представляют собой, как правило, встра-иваемые ЭВМ, функциональные возможности которых определяются решаемой задачей. При разработке программ для ПЛК традиционно используются языки программирования, специфицированные ГОСТ Р МЭК 61131-3-2016 [1].

На верхних уровнях (сетевое оборудование, серверы, диспетчерский интерфейс) обычно применяют оборудование и коммерческое программное обеспечение общего назначения, поскольку зачастую для организации доступа на этот уровень используется корпоративная сеть, связывающая все компоненты системы управления предприятием. Программное обеспечение для ЭВМ верхнего уровня разрабатывается на языках программирования общего назначения.

Для обмена данными с полевым обо-рудованием используются протоколы Modbus, OPC UA, МЭК 60870-5-101, МЭК 60870-5-104 и другие [2]. На верхних уровнях системы для обмена данными применяют протоколы TCP/IP.

Информация о параметрах и режимах функционирования АСУ ТП накапливается в базах данных для оперативной и последующей обработки.

С точки зрения архитектуры оборудования и программного обеспечения АСУ ТП аналогична «обычной» распределенной IT-системе, но при

этом обладает рядом особенностей, связанных с областью применения.

АСУ ТП является системой реального времени с критичным периодом реакции, при этом задержки и потеря данных неприемлемы, особенно в аварийных ситуациях. Компоненты системы могут быть расположены географически удаленно, в труднодоступных местах.

Перезагрузка АСУ ТП недопустима, а резервирование, особенно горячее, может потребоваться не только в случае аварии или сбоя, но и для выполнения плановых работ, по-скольку функционирование системы не должно прерываться. В этом состоит особенность про-изводственных систем, скажем, по сравнению банковскими – в них могут отключать на время часть функциональных возможностей, в том числе, для проведения, запланированных обновлений (обычно это происходит в ночное время).

Риски АСУ ТП в первую очередь связаны с физическими процессами. Безопасность людей, обусловленная безопасностью физических процессов, является первостепенной. Критически важна отказоустойчивость, и даже кратковременный простой в случае отказа должен быть исключен.

Поддержка и эксплуатация АСУ ТП осуществляется одним интегратором, который привлекает к работе поставщиков (разработчиков).

Жизненный цикл такой системы составляет 10-15 лет и более. Поскольку обновление программного обеспечения требует прохождения всего цикла тестирования и сертификации, при эксплуатации системы вынужденно исполь-зуется устаревшее программное обеспечение и оборудование.

Более широкий перечень отличий системы АСУ ТП от «обычной» информационной системы приведен в стандарте NIST SP 800-82 Guide to Industrial Control Systems [3].

Перечисленные выше особенности АСУ ТП влияют как на характер кибератак, так и на способы обеспечения безопасности объекта, управляемого с помощью этой системы, поскольку функциональная безопасность объекта управления оказывается тесно связанной с информационной безопасностью АСУ ТП. Поэтому для обеспечения кибербезопасности объекта в целом требуются комплексные решения.

Например, при сбое программного обеспечения IT-систему просто перезагружают, а при обнаружении уязвимостей – обновляют программное обеспечение. В случае АСУ ТП сбои ПО потенциально влекут неприемлемые последствия и требуют сложных мер предотвращения и парирования угроз, например, перехода в режим ручного управления технологическим процессом и горячего резервирования Обновление программного обеспечения может потребовать проведения полного цикла тестирования и сертификации.

Еще один пример особенности АСУ ТП касается регулярно обновляемых антивирусных программ, применение которых в целях обеспечения информационной безопасности в настоящее время является рутинной процедурой. Использование этой возможности в АСУ ТП, функционирующей в реальном масштабе времени, становится проблемой, поскольку дополнительно загружает вычислительные мощности и требует частой модификации ПО. Проблема, в частности, связана с ПЛК, которые не обладают достаточным быстродействием. В таком случае рекомендовано физически сегментировать сеть с помощью дополнительных устройств – межсетевых экранов, которые будут противодействовать внедрению зловредного ПО в соответствующий сегмент за счет настройки параметров передачи данных по сети, снимая таким образом дополнительную нагрузку с ПЛК.

2. Обеспечение кибербезопасности АСУ ТП

На протяжении последних десятилетий традиционными целями кибератак были и остаются домашние компьютеры, офисные и телекоммуникационные сети, системы поставщиков различных информационных услуг. Кибератаки на IT-системы носят массовый характер: в частности, сообщается, что системы «Лаборатории Касперского» автоматически исследуют и обрабатывают более 300 000 новых экземпляров подозрительного и вредоносного ПО ежедневно [8]. Информация об этом типе злонамеренных действий публична, по ним накоплена статистика, что помогает адекватно оценивать риски, а затем разрабатывать методы предотвращения и отражения угроз. В результате постепенно растет уровень защищенности, а действия кибермошенников в этой области становятся рискованным видом нелегальной деятельности, поэтому им приходится искать новые, менее защищенные цели.

В последнее время специалисты «Лаборатории Касперского» отмечают рост числа кибератак на компьютеры АСУ ТП. Согласно их отчету, во втором полугодии 2020 года по всему миру атакам подверглись до 40% компьютеров, входящих в состав технологической инфраструктуры предприятий или используемых для разработки систем автоматизации производства [9]. Кроме целевых атак, происходили и случайные инциденты, например, заражение техноло-

гических сетей зловредным ПО в результате использования устаревших программ (например, системы Windows XP) или из-за слабой защиты сети по причине некомпетентности или беспечности персонала и игнорирования угроз безопасности.

Как указано в отчете [9], во второй половине 2020 года кибермошенники-вымогатели ожидаемо интересовались платежеспособными жертвами, а основными источниками угроз для АСУ ТП оказались (в порядке убывания доли атакованных компьютеров) Интернет, съемные носители и почтовые клиенты. Масштабы кибератак характеризует доля атакованных компьютеров в некоторых отраслях: автоматизация зданий – 46,7%, нефтегазовая индустрия – 44%, инжиниринг и разработка АСУ – 39,3%, энергетика – 36,9%, машиностроение – 35%.

В технологически развитых странах проблема обеспечения киберзащиты АСУ ТП сейчас вышла на государственный уровень.

В России деятельность в этой области регулируется указами Президента РФ, федеральными законами, приказами Федеральной службы по техническому и экспортному контролю (ФСТЭК) и прочими регламентирующими документами. Ознакомиться с ними можно на сайте ФСТЭК.

Данные документы содержат требования по безопасности информационных технологий и защите информации, а также требования по безопасности объектов критической информационной инфраструктуры ([10], [11]).

Помимо законодательной базы применяются стандарты, предназначенные для использования при проектировании и разработке систем управления и охватывающие проблемы как функциональной, так и информационной безопасности (например, [4], [5], [6], [7]).

В последнее время количество стандартов растет лавинообразно и это, в частности, связано с тем, что области применения стандартов (отрасли промышленности) наделены своей спецификой, которую необходимо отражать в положениях стандарта и, соответственно, учитывать при разработке и эксплуатации АСУ ТП.

Рассмотрим требования, предъявляемые следующими стандартами: ГОСТ Р МЭК 61508 «Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью» [4] и ГОСТ Р МЭК 62443 «Сети промышленной коммуникации. Безопасность сетей и систем» [5]. Оценим соответствие ПЛК, функционирующего в составе аппаратно-программной платформы «Багет» разработки ФГУП ФНЦ НИИСИ РАН, требованиям стандарта ГОСТ Р МЭК 62443, предъявляемым к поставщикам и производителям компонентов АСУ ТП. В заключение рассмотрим ГОСТ Р 56939-2016 «Защита информации. Разработка безопасного программного обеспечения. Общие требования» [7], в котором применен иной подход к процедуре стандартизации.

3.1. Требования функциональной безопасности

Стандарт ГОСТ Р МЭК 61508 соответствует международному стандарту IEC 61508 «Functional safety of electrical/electronic/programmable electronic safety-related systems» и состоит из семи частей, которые охватывают все этапы жизненного цикла системы, начиная от проектирования и разработки (включая программное обеспечение) до ввода в эксплуатацию, техобслуживания и утилизации.

Разработчики данного стандарта видят свою задачу в том, чтобы установить общий подход к вопросам обеспечения функциональной безопасности для всех систем, состоящих из электрических и/или электронных, и/или программируемых электронных (Э/Э/ПЭ) элементов. Фактически это основа, которую предлагается использовать при разработке стандартов для отдельных видов продукции и областей применения.

В настоящее время уже существует целый ряд стандартов, регламентирующих вопросы функциональной безопасности: ГОСТ Р МЭК 61511 (системы безопасности для промыш-ленных процессов), ГОСТ Р МЭК 62061 (безопасность оборудования), ГОСТ Р МЭК 61513 (системы контроля и управления, важные для атомной станции) и др., разработанных на основе стандарта ГОСТ Р МЭК 61508.

Согласно стандарту ГОСТ Р МЭК 61508, под функциональной безопасностью подра-зумевается корректное функционирование как системы управления, так и управляемого ею оборудования, а при возникновении отказов – перевод объекта управления в безопасное состояние.

Этот стандарт основан на риск-ориентированном подходе, и его главная цель состоит в том, чтобы в первую очередь защи-тить персонал и окружающую среду от потен-циального вреда, а также поддерживать необ-ходимый уровень безопасности на протяжении всего цикла существования объекта управ-ления.

Стандарт вводит понятие уровня полноты безопасности, под которым подразумевается вероятность того, что система будет корректно выполнять функции обеспечения безопасности при всех заданных условиях в течение заданного интервала времени. Другими словами, чем меньше будет отказов в работе системы обеспечения безопасности (то есть, недиагностируемых опасных

отказов в работе объекта управления), тем надежнее и безо-паснее функционирование объекта управления. Руководство по определению уровня полноты безопасности с учетом приемлемого риска содержится в части 5 стандарта ГОСТ Р МЭК 61508.

Стандарт учитывает два типа отказов: случайные и систематические. Случайные отка-зы обусловлены выходом из строя аппаратных компонентов и требуют применения таких методов, как резервирование, самодиагностика, повышение устойчивости к внешним воздействиям и т.п. Оборудование также должно быть ремонтопригодным и долговечным.

Уровень полноты безопасности (УПБ) связан со случайными отказами и может иметь значения от 1 до 4. Для каждого уровня заданы значения показателей безопасности и требования к реализации процессов жизненного цикла (см. таблицу 1) либо в виде вероятности отказа (для редко вызываемых функций обеспечения безопасности, не более одного раза в год), либо в виде допустимой частоты отказа (для часто или непрерывно вызываемых функций обеспечения безопасности).

Таблица 1. Уровни полноты безопасности согласно стандарту МЭК 61508

Уровень полноты безопасности, требуемый для снижения вероятного ущерба	**PFDavg: средняя вероятность отказа выполнения по запросу (редкие вызовы)**	**PFH: средняя частота отказов в час (частые вызовы)**
УПБ 4 (техногенная катастрофа)	$>=10^{-5}$ -$<10^{-4}$	$>=10^{-9}$ -$<10^{-8}$
УПБ 3 (гибель персонала или населения)	$>=10^{-4}$ -$<10^{-3}$	$>=10^{-8}$ -$<10^{-7}$
УПБ 2 (травматизм персонала)	$>=10^{-3}$ -$<10^{-2}$	$>=10^{-7}$ -$<10^{-6}$
УПБ 1 (ущерб для оборудования и продукции)	$>=10^{-2}$ -$<10^{-1}$	$>=10^{-6}$ -$<10^{-5}$

Причиной систематических отказов, согласно стандарту, являются ошибки проектирования и эксплуатации, в том числе, и ошибки программного обеспечения. Систематические отказы аппаратуры также могут быть вызваны внешним воздействием (климатическим, радиационным, механическим и прочими).

Для устранения систематических отказов стандарт рекомендует совершенствовать процессы проектирования, разработки и тестирования, следовать методикам управления проектами, подробно документировать выполняемые действия, а также использовать апробированные и надежные инструментальные средства разработки (компиляторы, операционные системы и т.п.).

В частности, одним из обязательных условий при сертификации на соответствие стандарту МЭК 61508 в случае, если система связана с безопасностью, является соблюдение V-цикла разработки, указанного в части 61508-3 данного стандарта (рис. 2).

Часть 61508-7 стандарта МЭК 61508 содержит рекомендации по методам и средствам, используемым для достижения желаемого уровня безопасности физических устройств и программного обеспечения. Большинство методов являются комплексными, то есть, в той или иной степени они направлены на защиту как от случайных, так и от систематических отказов.

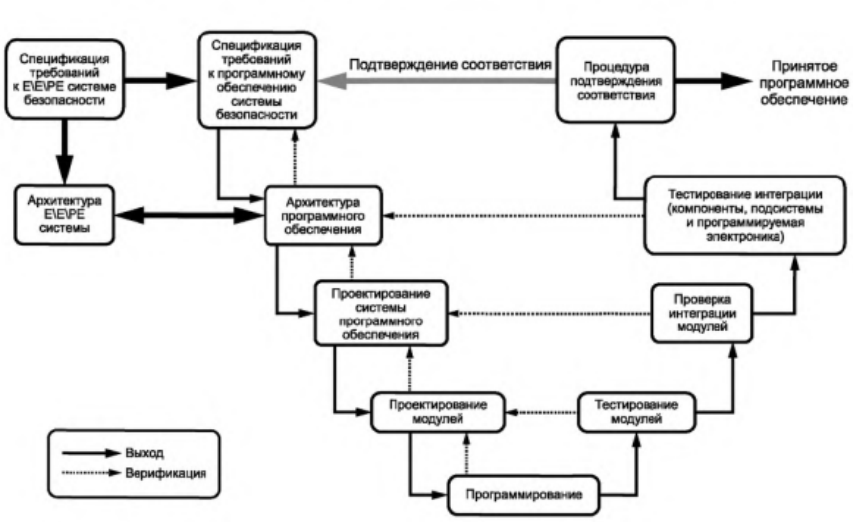

Рис. 2. Жизненный цикл разработки программного обеспечения (V-цикл; МЭК 61508-3, рисунок 6)

Для внедрения стандарта в ежедневную практику будет необходимо подробно раз-работать и соблюдать определенные рабочие процессы, которые в целом соответствуют по-ложениям управления проектами, программной и системной инженерии, но учитывают спе-цифику области применения и методы работы, принятые в конкретной компании.

3.2. Требования информационной безопасности

Задача обеспечения функциональной безопасности АСУ ТП тесно связана с инфор-мационной безопасностью системы, поскольку одновременно с широким внедрением циф-ровых методов в управление технологическими процессами возникли новые угрозы АСУ ТП, главным образом обусловленные исполь-зованием сетей:

- внешнее проникновение в систему с выведением АСУ ТП и управляемых объектов из строя;
- внешнее несанкционированное управление компонентами системы с определенными целями, в том числе, промышленный шпионаж;
- блокирование доступа к АСУ ТП и управляемым объектам;
- несанкционированное обновление ПО с целью изменения режимов работы технологических объектов.

АСУ ТП обычно имеет следующие уязви-мости: периметр сети, дистанционные подклю-чения, неправильно настроенные сетевые экра-ны, закладки в приобретаемом оборудовании, сменные носители информации (в первую оче-редь, использующие USB-порты), а также прог-раммное обеспечение и коммуникационные линии [9].

Как и в случае с функциональной безо-пасностью, теме информационной безо-пасности посвящено много различных стан-дартов и количество их растет. Среди них стан-дарт ГОСТ Р МЭК 62443 «Сети промышленной коммуникации. Безопасность сетей и систем», настоящая энциклопедия информационной безопасности АСУ ТП [5].

Стандарт состоит из четырех частей, каждая из которых включает несколько документов:

- общие положения (терминология и концепции);
- политики и регламенты (требования к системе управления безопасностью, руководство по внедрению этой системы, руководство по утилизации системы);
- системные требования (технологии безопасности, уровни безопасности);
- компонентные требования (технические требования безопасности компонентов, требования разработчиков продукции).

Стандарт задает требования к проек-тированию и эксплуатации систем безо-пасности, предназначенных для применения в уже существующих АСУ ТП («наложенных» систем), а также систем безопасности, вклю-чаемых в состав АСУ ТП при разработке. Согласно стандарту,

безопасность рассмат-ривается как совокупность связанных между собой непрерывных процессов, которые необ-ходимо поддерживать на всех стадиях жиз-ненного цикла системы.

По аналогии с функциональной безопас-ностью используется риск-ориентированный подход и понятие уровня безопасности.

Проектирование системы управления безопасностью стандарт рекомендует начать с оценки рисков, чтобы определить последствия, касающиеся финансов, здоровья и безопасности людей, окружающей среды, социальной, по-литической, экономической, военной или иной области, в случае нарушения доступности, целостности или конфиденциальности АСУ ТП. Затем следует выполнить анализ объекта защиты.

Стандарт предлагает несколько этапов анализа и моделирования объекта защиты. На первом этапе строится референсная модель, включающая пять логических уровней:

- уровень 4: управление предприятием;
- уровень 3: операционное управление производством;
- уровень 2: управление и мониторинг технологических процессов (SCADA);
- уровень 1: локальное управление процессом и оборудованием;
- уровень 0: физический процесс и оборудование (датчики и исполнительные механизмы).

Типичная АСУ ТП соответствует уровням 0, 1 и 2.

На следующем этапе строится объектная модель, которая описывает иерархию основных объектов, взаимодействие с сетями, территориальными площадками, ключевыми подразделениями, системами контроля и прочим технологическим оборудованием. Данная модель позволяет наглядно представить критические процессы и активы.

Далее предлагается построить модель архитектуры, которая включает все основные элементы АСУ ТП, телекоммуникационное оборудование, линии связи и т.д. Данная модель будет использована для зонирования. Зоны формируются с учетом уровня риска, функциональных/технических характеристик, логических или физических границ, используемых сетей передачи данных и т.п. Наличие выделенных зон облегчает задачу выбора и реализации контрмер для снижения рисков.

Для каждой зоны определяется текущий уровень безопасности. Данное понятие описывает реализацию требований к мерам безопасности по семи основным направлениям:

- идентификация и аутентификация,
- контроль использования ресурсов,
- целостность системы,
- конфиденциальность информации,
- управление информационными потоками,
- управление событиями (время реакции на события),
- доступность ресурсов.

Каждое направление содержит требования, комбинации которых определяют уровень безопасности.

Используют четыре уровня безопасности:

- 0: меры обеспечения безопасности не требуются (с учетом разумной достаточности; обычно это касается отдельных компонентов, например, датчиков);
- 1: требуется защита от случайных нарушений (обеспечивается с помощью установленных процедур);
- 2: требуется защита от злонамеренных нарушений, например, вирусов или известных уязвимостей (как правило, такие атаки отражаются автоматически);
- 3: требуется защита от злоумышленников, обладающих достаточными знаниями и ресурсами, чтобы совершить атаку на целевую систему (например, с использованием программных инструментов, которые требуют специальных знаний);
- 4: требуется защита от злоумышленников, располагающих значительными ресурсами, например, множеством мощных компьютеров в течение длительного времени.

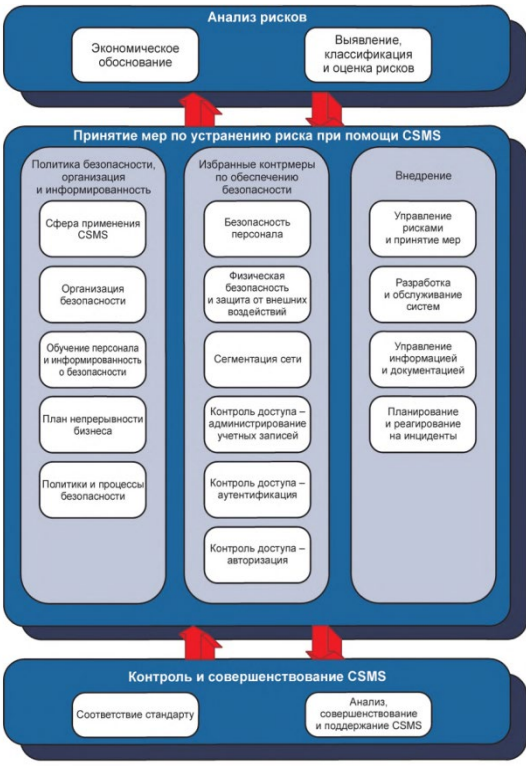

Рис. 3 Структура элементов системы управления кибербезопасностью (ГОСТ Р МЭК 62443-2-1)

Структура элементов предлагаемой стан-дартом системы управления кибербезопас-ностью (CSMS, Cyber Security Management System) показана на рис. 3.

Однако стандарт ГОСТ Р МЭК 62443, как и ГОСТ Р МЭК 61508, содержит «обобщенные» формулировки и не предназначен для того, чтобы описывать последовательный процесс создания системы, включающей данные элементы. Каждая организация должна самостоятельно создать такой процесс в соответствии с принятой культурой, структурой и действующим состоянием работ по обеспечению кибербезопасности.

3.3. Оценка соответствия ПЛК, функционирующего в составе платформы «Багет», требованиям стандарта ГОСТ Р МЭК 62443

В заключение перечислим требования, выдвигаемые стандартом ГОСТ Р МЭК 62443 к производителям и поставщикам компонентов АСУ ТП (часть 62443-4-2), а также оценим степень соответствия ПЛК, функционирующего в составе аппаратно-программной платформы «Багет» разработки ФГУП ФНЦ НИИСИ РАН, требованиям уровня безопасности 1 и 2 (в тексте стандарта обозначены SL 1, SL 2). Требования касаются только разработчиков и обслуживающего персонала и соответствуют семи основным направлениям обеспечения безопасности, перечисленным выше.

Архитектура аппаратно-программной платформы «Багет» и способы ее использования описаны в [12].

Характерной особенностью программируемых логических контроллеров семейства «Багет» является использование технологии кросс-разработки для создания и установки на ПЛК специального программного обеспечения (СПО). СПО компонуется с модулями операционной системы, в результате чего формируется исполняемый образ, записываемый в энергонезависимую память в процессе изготовления ПЛК.

При подаче питания стартует программа, записанная в ПЗУ ПЛК при изготовлении процессорного модуля. Данная программа передает загружает исполняемый образ в ОЗУ ПЛК и передает ему управление. Для внесения изменений в программу ПЗУ и исполняемый образ необходимо аппаратное воздействие на ПЛК, поэтому при обеспечении соответствующего уровня защиты устройства от физического воздействия состав СПО можно считать предопределенным. Взаимодействие СПО с сетевым окружением происходит только в той мере, в какой это определено алгоритмом функционирования СПО.

Требования идентификации и аутентификации. Если компонент сети дает пользователям возможность получения доступа к устройствам или приложениям, то этот сетевой компонент должен иметь возможность однозначной идентификации и аутентификации всех поль-зователей, включая людей, процессы и устройства.

ПЛК в составе аппаратно-программной платформы «Багет» не выполняет аутентификацию пользователей и не содержит необходимой для этого информации, поскольку эта функция перенесена на более высокий уровень и внешние по отношению к ПЛК средства (SCADA-станции, диагностические ЭВМ). Однако ПЛК обеспечивает разграничение доступа для аутентифицированных пользователей (например, операторов SCADA-станций) с помощью базы учетных записей и идентификаторов, как описано в [12].

Порядок использования сетевого окружения полностью определен алгоритмом СПО, а идентификацию каналов передачи данных и подключенных устройств ПЛК «Багет» выполняет с использованием протоколов передачи данных в момент включения питания.

Угроза подключения нарушителя к каналу передачи данных применима, но для этого требуется физическое вторжение в сеть, поскольку беспроводные каналы не используются. Невозможно также подключение нелегальных дополнительных устройств, так как их использование

не предусмотрено алгоритмом СПО, исполняемого на ПЛК.

Для реализации угрозы требуется нарушитель с высоким потенциалом, вероятность реализации угрозы минимальна.

Для разграничения доступа допускается использование паролей. Информация о неудачных попытках подключения к системе и об использовании системы направляется на диагностическую ЭВМ. Угроза отключения диагностической ЭВМ применима, для реализации угрозы требуется нарушитель со средним потенциалом.

Доступ через недоверенные сети контролируется внешними по отношению к ПЛК средствами, например, сетевыми экранами.

Требования контроля использования. Возможность управления учетными записями, включая операции по их созданию, активации, изменению, отключению и удалению, должна поддерживаться по всей сети. Это гарантирует, что никакие учетные записи не создаются, не изменяются и не удаляются, если не было предоставлено разрешение, а также запрещает устройствам устанавливать любые неавторизованные соединения.

ПЛК «Багет» обеспечивает разграничение доступа для аутентифицированных пользователей (например, операторов SCADA-станций) с помощью базы учетных записей и идентификаторов, например, так, как описано в [12]. Несанкционированное изменение учетной записи возможно, но угроза технологически трудно реализуема, так как для этого необходимо внесение изменений в программу, исполняемую на ПЛК, что требует действий внутреннего нарушителя с высоким потенциалом.

Подключение беспроводных каналов невозможно, поскольку их использование не предусмотрено в программе, исполняемой на ПЛК. Подключение нелегальных устройств возможно, но требует физического вторжения в сеть. Использование мобильного кода (Java, JavaScript, ActiveX, PDF, Postscript, Shockwave, Flash-анимации, VBScript) на ПЛК «Багет» технически не предусмотрено.

Блокировка сеанса работы может быть реализована на ПЛК, если это допускает технологический процесс, управляемый с помощью АСУ.

Процедура регистрации событий, под-лежащих аудиту, реализуется с помощью диаг-ностической ЭВМ. Накопитель данных явля-ется внешним по отношению к ПЛК «Багет», и его емкость является практически неог-раниченной. Обработка данных аудита также выполняется на внешних устройствах. Угроза отключения диагностической ЭВМ применима, для реализации угрозы требуется нарушитель со средним потенциалом.

Требования целостности системы. Любой компонент сети, имеющий пользовательский интерфейс, должен напрямую интегрироваться в систему, которая идентифицирует людей по пользовательскому, групповому, ролевому и/или системному интерфейсу. Это лишает пользователей возможности подключения к сетевым устройствам, доступ к которым им не предоставлен.

Угроза несанкционированного доступа к активному или пассивному физическому сетевому оборудованию из физической сети применима. Данная угроза относится к сети в целом. Парирование возможно с помощью межсетевых экранов или путем защиты периметра сетевого оборудования.

Угроза внедрения вредоносного кода на ПЛК «Багет» применима, но нарушение целостности программного обеспечения и информации возможно только при вмешательстве внутреннего нарушителя с высоким потенциалом путем внесения изменений в программу, исполняемую на ПЛК. Вероятность реализации угрозы минимальна.

Для верификации функциональной безопасности при начальном включении ПЛК «Багет» используются специальные тесты. Возможности применения таких тестов при штатном функционировании системы зависят от параметров технологического процесса.

Валидация входных данных и обеспечение детерминированного потока выходных данных должны быть реализованы с помощью программ, исполняемых на ПЛК. Обработка ошибок и формирование сообщений реализуется с помощью программ, исполняемых на ПЛК.

Требования конфиденциальности. Все устройства в сети должны иметь возможность подтверждать достоверность любых запросов на обновление системы/встроенного программного обеспечения и проверять, что источник не пытается загрузить какие-либо вирусы или вредоносные программы. Это может быть реализовано за счет использования токенов, ключей, сертификатов или паролей.

Защита конфиденциальности информации, хранящейся на ПЛК «Багет», осуществляется внешними по отношению к ПЛК средствами (SCADA-станции, диагностические ЭВМ). В случае необходимости возможно применение криптографии.

Угроза загрузки вируса или вредоносной программы при обновлении встроенного программного обеспечения применима, но для этого требуется физический доступ к ПЛК внутреннего нарушителя с высоким потенциалом. Вероятность реализации угрозы минимальна.

Требования ограничения потока данных. Для

создания безопасных соединений следует использовать аутентификацию на основе открытого ключа, поскольку она предотвращает ошибочную адресацию информации, а также предотвращает передачу конфиденциальной информации, которая должна оставаться в сети, в неподтвержденные внешние источники запросов.

Принцип построения и конфигурация аппаратно-программной платформы «Багет» исключает возможность несанкционированного подключения в штатном режиме каких-либо каналов передачи информации за исключением тех, использование которых предусмотрено в программе ПЛК, поэтому аутентификация соединений на ПЛК не выполняется.

Защита границ зоны выполняется внешними по отношению к ПЛК средствами (SCADA-станции, диагностические ЭВМ).

На ПЛК вне системы отсутствует информация, которая могла бы быть передана в режиме абонент – абонент (электронная почта, Твиттер, Фейсбук, исполнимые файлы).

Требования по управлению доступностью ресурсов. Все устройства, которые появляются в сети, должны поддерживать аутентификацию при входе в систему. Чтобы запретить нежелательным пользователям доступ к устройству или сети, приложение или устройство должны ограничивать количество попыток ввода пользователем неправильного пароля перед блокировкой.

Защиту ПЛК «Багет» от событий DoS выполняют внешние по отношению к ПЛК средства (SCADA-станции, диагностические ЭВМ).

Задачи ограничения использования ресурсов, резервирования, восстановления после сбоя, аварийного питания и минимальной функциональности могут быть реализованы на ПЛК «Багет» в соответствии с требуемыми алгоритмами автоматизированного управления.

Конфигурирование сети выполняется в технологическом режиме функционирования ПЛК «Багет», и программа ПЛК в штатном режиме работает в рамках установленной конфигурации.

Таким образом, ПЛК как компонент в составе АСУ ТП на основе аппаратно-программной платформы «Багет» реализует требования стандарта ГОСТ Р МЭК 62443 на уровне SL 1 – SL 2 при соблюдении организаци-онных мер, касающихся контроля физического доступа к устройствам и коммуникациям.

3.4. Новые стандарты разработки безопасного программного обеспечения

Применение стандартов, содержащих только требования в обобщенной, лишенной кон-кретности формулировке, порождает неод-нозначность при выстраивании процедур раз-работки программных систем и их серти-фикации. В настоящее время в Российской федерации проходит процедуру утверждения линейка новых стандартов ГОСТ Р «Защита информации. Разработка безопасного про-граммного обеспечения». Данный комплект стандартов будет содержать не только перечень требуемых мер, но также рекомендации по их реализации и методику проверки соответствия стандарту сертифицируемого ПО.

При подготовке новых стандартов был проведен анализ существующих механизмов, направленных на уменьшение числа уязвимостей в разрабатываемом ПО, и был сформирован базовый набор требований, позволяющий провести оценку соответствия процессов разработки ПО данным требованиям [13].

Новый комплект включает следующие части [14]:

– ГОСТ Р 56939-2016 «Защита информации. Разработка безопасного программного обеспечения. Общие требования» [7];

– ГОСТ Р «Защита информации. Разработка безопасного программного обеспечения. Руководство по оценке безопасности разработки программного обеспечения»;

– ГОСТ Р «Защита информации. Разработка безопасного программного обеспечения. Руководство по проведению статического анализа. Общие требования»;

– ГОСТ Р «Защита информации. Разработка безопасного программного обеспечения. Руководство по проведению динамического анализа. Общие требования»;

– ГОСТ Р «Защита информации. Разработка безопасного программного обеспечения. Доверенный компилятор языков Си/Си++. Общие требования»;

– ГОСТ Р «Защита информации. Разработка безопасного программного обеспечения. Управление безопасностью программного обеспечения при использовании заимствованных и привлекаемых компонентов».

Часть «Общие требования» утверждена и введена в действие. Также с 1 июня 2019 года применяется «Методика выявления уязви-мо-

стей и недекларированных возможностей в программном обеспечении», утвержденная ФСТЭК.

ГОСТ Р 56939-2016 содержит 28 элементов, соответствующих требуемым мерам, в сле-дующем формате: описание меры, реко-мендации по ее реализации, перечень до-кументации, распределение ролей и обязан-ностей. Перечисленные меры охватывают весь процесс создания ПО, начиная от анализа требований и заканчивая решением проблем в процессе эксплуатации, а также включает меры менеджмента документацией, инфраструктурой среды разработки и людскими ресурсами.

Завершение работы над комплектом стандартов запланировано на 2021 год и еще не завершено.

4. Заключение

В настоящее время в распоряжении разработчиков и пользователей IT-систем потенциально имеется широкий набор средств обеспечения кибербезопасности. Решение этой задачи дается непросто и требует больших затрат, но не стоит опускать руки. В ближайшем будущем «Лаборатория Касперского» прогнозирует следующие события [15].

Атаки на промышленные предприятия, АСУ ТП и полевые устройства станут более целенаправленными. Снятие с поддержки Widows 7 и Server 2008, которые широко применяются в промышленных сетях, усугубит ситуацию.

Увеличится число атак с вымоганием денег под угрозой публикации или продажи украденных документов.

Более активным станет кибершпионаж. Возрастет число APT (advanced persistent threat)-группировок, то есть, группировок, создающих постоянные серьезные угрозы, обладающих специальными знаниями и значительными ресурсами.

Переход в онлайн и цифровизация муниципальных и государственных сервисов сделают их более уязвимыми для злоумышленного дистанционного доступа, создадут больше возможностей для кросс-ведомственных атак и атак на смежные структуры.

Киберугрозы для промышленных предприятий, в том числе, для объектов критически важной инфраструктуры стали реалиями сегодняшнего дня.

Способы создания надежного программного обеспечения и безопасной среды функционирования подробно регламентируются законами, стандартами, другими документами и постоянно совершенствуются. Необходимо последовательно внедрять их в ежедневную практику, несмотря на значительную трудоемкость требуемых процедур.

Это надежный способ обеспечения кибербезопасности на сегодняшний день.

Cybersecurity Issues in the Development and Use of ICS

Andrey Gryuntal, Svetlana Bazaeva

Abstract. The article contains an overview of the basic requirements and methods used to ensure the cybersecurity of technological processes controlled by an industrial control system. Using the example of a PLC operating as part of the "Baguette" hardware/software platform developed by Federal scientific center research Institute of system studies of the Russian Academy of Sciences, an assessment of the degree of compliance with the requirements put forward by the GOST R IEC 62443 standard for manufacturers and suppliers of ICS components is given.

Keywords: ICS, cybersecurity, PLC, "Baguette" platform

Литература

1. ГОСТ Р МЭК 61131-3-2016 Контроллеры программируемые. Часть 3. Языки программирования.

2. ГОСТ Р МЭК 60870-5. Устройства и системы телемеханики. Часть 5. Протоколы передачи.

3. NIST SP 800-82. Guide to industrial control systems (ICS) security.

4. ГОСТ Р МЭК 61508. Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью.

5. ГОСТ Р МЭК 62443. Сети промышленной коммуникации. Безопасность сетей и систем.

6. ГОСТ Р ИСО/МЭК 27000. Информационная технология. Методы и средства обеспечения безопасности. Системы менеджмента информационной безопасности. Общий обзор и терминология.

7. ГОСТ Р 56939-2016. Защита информации. Разработка безопасного программного обеспечения. Общие требования.

8. Е. Гончаров. Проблемы киберзащиты промышленных предприятий. «Лаборатория Касперского», 2018, ics-cert.kaspersky.ru/reports/2018/12/05/challenges-of-industrial-cybersecurity/.

9. Ландшафт угроз для систем промышленной автоматизации. Статистика за второе полугодие 2020. «Лаборатория Касперского», 2021, ics-cert.kaspersky.ru/reports/2021/03/25/threat-landscape-for-industrial-automation-systems-statistics-for-h2-2020/.

10. Документы по обеспечению безопасности критической информационной инфраструктуры. ФСТЭК России, fstec.ru/tekhnicheskaya-zashchita-informatsii/obespechenie-bezopasnosti-kriticheskoj-informatsionnoj-infrastruktury/285-zakony.

11. Нормативные правовые акты, организационно-распорядительные документы, нормативные и методические документы и подготовленные проекты документов по технической защите информации. ФСТЭК России, fstec.ru/tekhnicheskaya-zashchita-informatsii/dokumenty/107-zakony.

12. А.И. Грюнталь, С.Г. Дышленко. Разграничение доступа в автоматизированных системах, функционирующих под управлением операционных систем семейства «Багет». Программная инженерия, Т. 10, № 3, 99-104, 2019 г.

13. А.В. Барабанов, А.С. Марков, В.Л. Цирлов. 28 магических мер разработки безопасного программного обеспечения. Вопросы кибербезопасности, №5(13)-2015.

14. В.А. Падарян. Разработка доверенного ПО. Вызовы и перспективы. Форум «Технологии безопасности» Конференция «Актуальные вопросы защиты информации», 11.02.2021, www.tbforum.ru.

15. Е. Гончаров Е. Киберугрозы для промышленных предприятий в 2021 году. «Лаборатория Касперского», 2020, securelist.ru/ics-threat-predictions-for-2021/99417/.

Визуальная интегрированная среда как инструмент для разработки программ с критической миссией

К.Г. Нархов[1], Я.А. Зотов[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия kostas@niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, zotov@niisi.ras.ru

Аннотация. В статье рассматривается визуальная интегрированная среда разработки специального программного обеспечения. Рассмотрены средства повышения отказоустойчивости приложений и автоматизации разработки приложений реального времени, в том числе генерация исходных текстов программ. Описано использование унифицированного языка моделирования (UML) и расширяемого языка разметки (XML) для проектирования приложений реального времени. Рассмотрены подходы описания синтаксиса языков программирования в терминах XML.

Ключевые слова: программа реального времени, автоматизация разработки, генерация кода, формализация языка программирования

1. Введение

К специальному программному обеспечению, выполняющемуся в реальном масштабе времени, предъявляются жесткие требования с точки зрения надежности и отказоустойчивости. Сокращение уязвимостей и ошибок, результатом которых может стать частичный или полный отказ программы, является трудоемкой задачей, для автоматизации которой была разработана визуальная интегрированная среда ТСАГ СПО (Технологические средства автоматизированной генерации специального программного обеспечения). ТСАГ СПО позволяют сокращать количество уязвимостей при разработке СПО реального времени за счет применения типовых программных компонентов. Типовые программные компоненты входят в состав поставляемой вместе с ТСАГ СПО библиотеки, к ним относятся синтаксические конструкции языка, утверждения и выражения функций ОС РВ Багет и шаблоны программ реального времени.

С помощью высокоуровневых визуальных средств программист может проектировать в ТСАГ СПО программы, в которых будет автоматизирована проверка исходных текстов на соответствие синтаксису целевого языка программирования и правильное использование функций ОС РВ Багет, а также будет предоставлена возможность использования шаблонов программ реального времени, в которых устранены типовые уязвимости.

Перечисленные средства позволяют сократить количество типовых уязвимостей и ошибок в программном обеспечении: синтаксических, семантических, ошибок синхронизации и ошибок данных.

2. Безопасность разработки программ для систем реального времени

Увеличение сложности вычислительных систем реального времени подняло на новый уровень проблему обеспечения их информационной безопасности. В настоящий момент требования к механизмам информационной безопасности стали обязательными функциональными требованиями, предъявляемыми к приложениям реального времени еще на этапе постановки задачи по их проектированию. При этом для систем жесткого реального времени всегда были и остаются особо актуальными вопросы функциональной безопасности – способности вычислительной системы не создавать угрозы жизни и здоровью людей или сохранности материальных активов. Таким образом, надежность и отказоустойчивость являются важными аспектами информационной безопасности программного обеспечения реального времени [4].

Одна из основных задач разработки программ реального времени – это обеспечение эффективности и надежности приложений уже на этапе проектирования. Безусловно, это требует высокого профессионального уровня разработчиков и наличия соответствующих инструментов, например среды разработки, отладки и моделирования приложений. Эти инструменты должны учитывать требования стандартов (ГОСТ Р ИСО/МЭК 1540, ГОСТ Р 51904-2002 и

т. д.), в частности, классификации категорий отказов, а также вытекающих из нее классификации уровней программного обеспечения и требований к тестированию этого программного обеспечения [5].

Следует отметить, что одним из направлений по повышению отказоустойчивости приложений является применение инструментов проектирования на основе современных инноваций – унифицированного языка моделирования (UML) [1] и расширяемого языка разметки (XML) [2]. Использование перечисленных технологий позволяет повысить надежность программ за счет применения типовых программных компонентов, а также уменьшить трудоемкость разработки.

3. Средства автоматизации разработки приложений реального времени

Повышение надежности приложений, работающих в режиме жесткого реального времени, может быть достигнуто применением в процессе создания приложений реального времени автоматизированных средств построения и анализа моделей вычислительных систем реального времени (ВС РВ), автоматизированной генерации прототипов типовых программных модулей на языке программирования (например на Си) для компонентов ВС РВ, имитационного моделирования работы ВС РВ.

Автоматизированные средства построения и анализа моделей представляют собой гибридный XML/UML редактор, с помощью которого возможно как создание UML диаграмм, так и специфицирование узлов и элементов этих диаграмм с помощью XML документов. Документы, описывающие диаграмму на новом уровне абстракции, должны удовлетворять некоторой XML схеме. По этой схеме можно провести проверку (валидацию) диаграммы, и уже на этапе проектирования иметь информацию о синтаксической корректности будущей программы.

Особое место занимает переход от визуального представления программы к исходному коду на алгоритмическом языке. Для решения данной задачи используется технология XML.

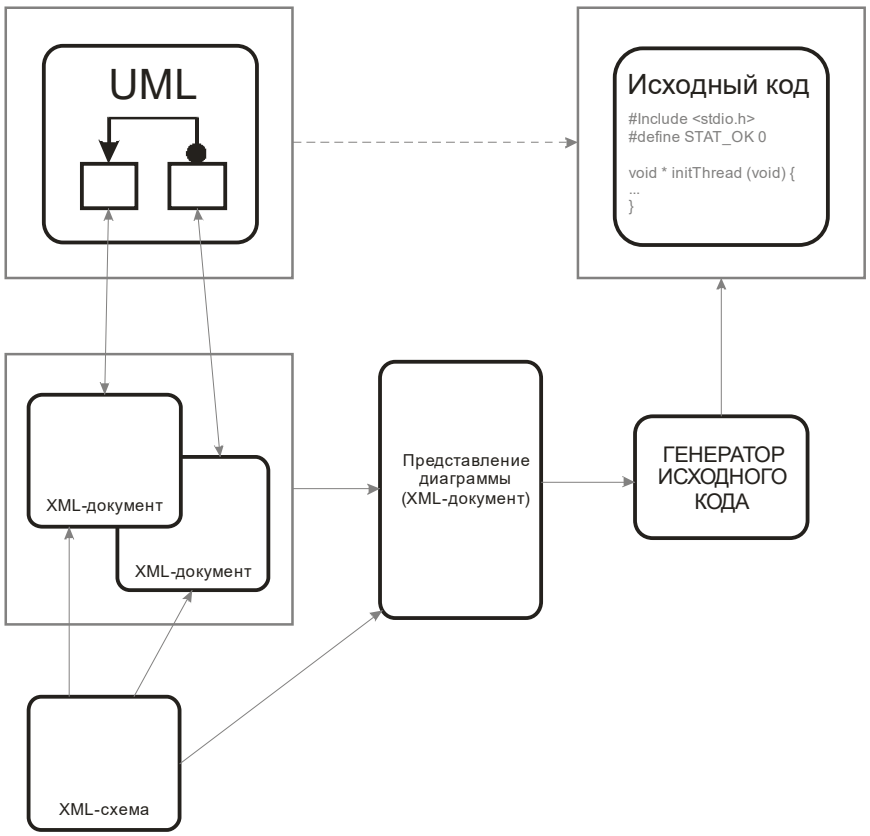

Рис. 1 – Принципиальная схема средств разработки

С ее помощью формализуется синтаксис некоторого алгоритмического языка и составляется соответствующая XML схема. В дальнейшем на основании этой схемы создаются базо-

вые конструкции, которые впоследствии используются в качестве представлений UML диаграмм.

Проектировщику предоставляется возможность «программировать» визуальными средствами (диаграммы), при этом исходный код, стоящий за тем или иным узлом диаграммы, является выверенным и предоставляется в качестве библиотеки конструкций (возможно даже типовых скелетов программ) к инструментарию проектирования.

Принципиальная схема средств разработки, основанная на этом подходе, представлена на рисунке 1.

4. Формализация языка программирования в терминах XML

Здесь под формализацией языка программирования понимается описание синтаксиса этого языка в терминах XML. В статье рассматривается два подхода к формализации языка, а именно: составление формального представления произвольного множества конструкций языка и составление формального представления на основании стандарта на язык программирования.

Формальное представление произвольного множества конструкций алгоритмического языка основано на выделении из множества его конструкций некоторого подмножества, составлении иерархии элементов этого подмножества, а также спецификации отношений (связей) между этими элементами. Например, для любого языка программирования можно выделить синтаксические конструкции «программа», «операция», «условный переход», «цикл», «вызов подпрограммы». Такое формальное представление имеет ряд преимуществ по сравнению с полным формальным представлением языка в соответствии со стандартом. Во-первых, оно является минимально достаточным с точки зрения результатов моделирования, т. к. в блоки выделяются только требуемые элементы программы. Во-вторых, программы, используемые для обработки формального представления произвольного множества конструкций языка, являются простыми в силу того, что подмножество формализованных элементов обычно намного меньше множества конструкций алгоритмического языка. Тем не менее, формальное представление произвольного множества конструкций языка не предоставляет в полном объеме возможностей для контроля синтаксических ошибок внешними средствами. Более того, модели программ, основанные на таком формальном представлении, являются ограниченными вследствие ограниченности самого формального представления.

Представление синтаксиса Си в терминах XML на основании стандарта выполняется путем полного описания синтаксиса алгоритмического языка Си, изложенного в стандарте [3].

Представление на основе стандарта является полным, следовательно, может быть использовано для представления любой Си-программы, кроме того, такое представление языка используется в средствах контроля синтаксических ошибок.

Выбор подхода к формализации не является однозначным и напрямую зависит от требований, предъявленных к формальному представлению. Отметим, что формальное представление произвольного множества конструкций алгоритмического языка удобно использовать, например, при имитационном моделировании. В этом случае с каждым элементом модели должны быть ассоциированы те или иные расчетные характеристики: быстродействие, объем требуемой оперативной памяти, интервал процессорного времени и т. д.

Формальное представление на основе стандарта языка программирования Си целесообразно к использованию в системах, где одними из важных требований являются проектирование произвольных отдельных участков исходного кода, контроль синтаксических ошибок и реинженеринг готовых программ.

5. Генератор исходного кода

В состав ТСАГ СПО входит генератор исходного кода. Генератор разработан в ФГУ ФНЦ НИИСИ РАН и прошел соответствующие сертификационные испытания, необходимые для допуска к использованию при разработки специального программного обеспечения. Он работает по событийно-поточному принципу разбора документов. Использование стратегии генерации исходного кода, основанной на потоках, подразумевает непрерывную передачу XML данных генератору. Генератор функционирует подобно каналу, в качестве исходных данных для которого выступает XML-разметка. Этот канал называется потоком событий. Происхождение подобного названия связано с тем, что каждый фрагмент анализируемых данных сигнализирует о некотором событии, происшедшем в потоке входных данных. Например, к важным событиям можно отнести начало нового элемента. В этом случае выполняется поиск инструкций по обработке, которые могут содержаться либо в самой разметке, либо во внешнем хранилище данных (например, в базе данных). В результате вы-

полнения каждого нового обновления потока событий (чтение XML документа, прием информации из сокета и т. д.), генератор выполняет новую трансляцию данных (смещение блоков данных в канале). Затем выполняется проверка текущего блока данных на предмет наличия специфического содержимого, которое передается соответствующему обработчику и в последующем помещается во внутренние информационные структуры (например, дерево представления входного XML документа или стек событий). Принципиальная схема генератора исходного кода представлена на рисунке 2.

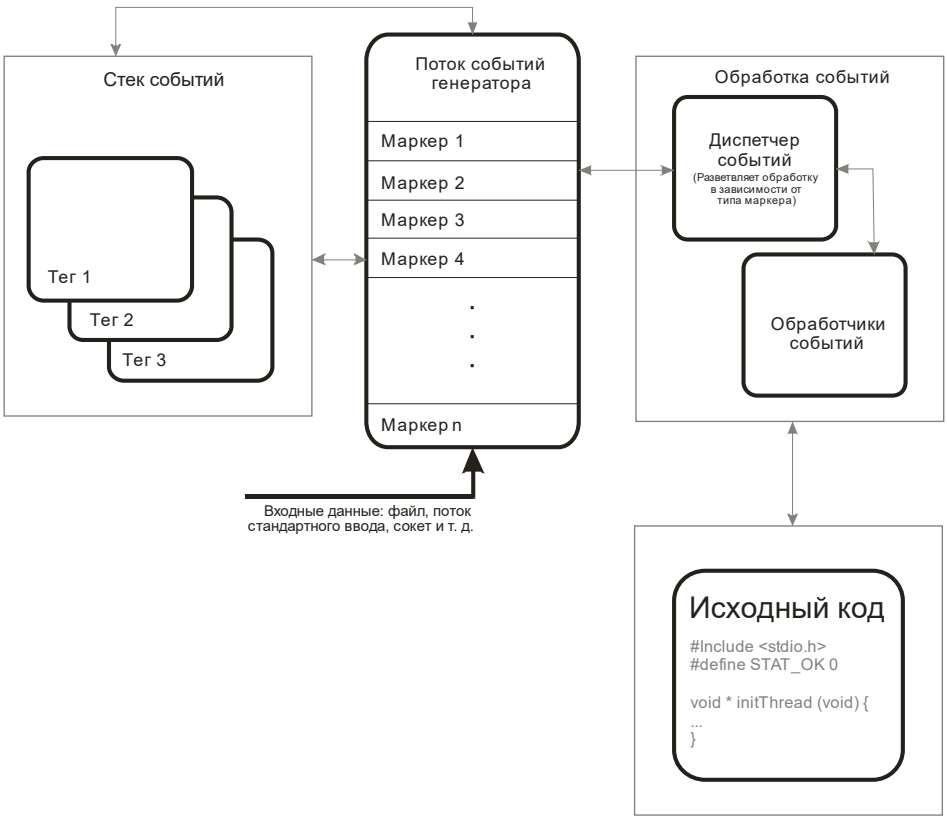

Рис. 2 – Принципиальная схема генератора исходного кода

Потоки событий генератора являются группами данных. Каждая группа данных, именуемая маркером, представляет собой набор из одного или большего количества символов. Каждый маркер характеризуется специальной разметкой, например, некоторым XML тегом. Благодаря этому упрощается разбор XML-кода, а также минимизируются требуемые ресурсы и время.

Особенность потоков заключается в содержимом, присущем каждому маркеру; маркер должен содержать XML код, который отвечал бы требованиям формальной корректности. Данные, которые не подчиняются требованиям формальной корректности, будут исключены из обработки генератором.

Язык XML иерархичен по своей структуре (элементы содержат другие элементы), в связи с этим невозможно выделять отдельные элементы и использовать их в качестве маркеров потока. В документе, отвечающем правилам формальной корректности, все элементы содержатся в одном корневом элементе. Корневой элемент, содержащий весь документ, сам по себе не является потоком. Поэтому трудно представить поток, формирующий завершенный элемент в виде маркера (если речь не идет о пустом элементе).

В силу упомянутых причин XML-потоки составляются на основе событий. Любое событие является сигналом, свидетельствующим об изменении состояния документа. Например, если генератор обнаруживает начальный тег элемента, сообщается, что был открыт другой элемент, и изменилось состояние синтаксического разбора. Завершающий тег влияет на состояние, закрывая последний открытый элемент. Обработчики событий XML могут отслеживать открытые элементы в структуре стека данных. При этом в стек помещаются сведения о только что открытых элементах и исключаются данные о закрытых элементах. В любой момент синтаксического разбора генератор и соответствующие обработчики могут оценить глубину своего нахождения в документе путем анализа стека.

После обработки данные помещаются в информационные структуры, которые в совокупности являются внутренним представлением XML документа. Эти структуры могут быть как деревьями, так и прочими представлениями данных (массивы, списки, очереди). Основные критерии выбора структуры для внутреннего представления – сохранение логических сущностей и связей между ними, а также удобство анализа и обработки. На основании внутреннего представления выполняется генерация исходных текстов программ на алгоритмическом языке.

Основные требования к генератору исходных текстов:

– генератор исходных текстов программ должен генерировать программные модули, содержащие прототипы процессов и потоков управления, пригодные для последующей компиляции, сборки и выполнения на аппаратных средствах проектируемой системы. Генерация исходных текстов программ должна выполняться на основе типовых заготовок таких программ с возможным интерактивным вмешательством программиста;

– генератор исходных текстов программ должен обеспечивать встраивание в генерируемые программы дополнительных операторов (на основе типовых заготовок и/или встраиваемых в интерактивном режиме), обеспечивающих протоколирование работы макета системы и имитацию поведения реальных прикладных программ при прогоне макета на имитационной модели.

6. Заключение

В ходе работы над визуальной интегральной средой разработки программного обеспечения ТСАГ СПО были разработаны автоматизированные средства построения и анализа моделей, библиотека типовых программных компонентов, генератор исходных текстов программ. Также было создано формализованное представление языка программирования Си в терминах XML.

Генератор исходных текстов может применяться как в виде отельного полнофункционального приложения, так и в составе системы проектирования. В настоящее время активно развиваются приложения, предоставляющие интерфейсы для разработки облачных расширений и плагинов. Таким образом, генератор исходных текстов может быть встроен в среды типа Eclipse или IntelliJ IDEA. Не менее важна перспектива создания XML библиотек конструкций для популярных языков типа Go, Java, Kotlin, а также реализация средств автоматизированного документирования сгенерированного исходного кода.

Особое место занимает подготовка скелетов программ, которые могут быть включены в XML библиотеку, поставляемую в составе ТСАГ СПО. Это могут быть скелеты потоков управления, процессов, критические участки кода, обработчики исключений и прерываний.

The Visual Integrated Development Environment for Critical Software

Konstantin Narkhov, Yaroslav Zotov

Abstract. The article describes the visual integrated development environment for special software. The ways of improving the fault tolerance of applications and real-time applications development automation, including the program source codes generation, are considered. The usage of Unified Modeling Language (UML) and Extensible Markup Language (XML) for creating real-time applications is described. The approaches to the syntax description of programming languages in XML terms are considered.

Keywords: realtime software, development automation, code generation, programming language formalization

Литература

1. ISO/IEC 19501:2005(E), January 2005. Unified Modeling Language Specification. Version 1.4.2 formal/05-04-01.

2. Extensible Markup Language (XML) 1.0 (Fourth Edition) W3C Recommendation 16 August 2006, edited in place 29 September 2006, (http://www.w3.org/TR/REC-xml/, дата обращения: 16.09.2010).

3. ISO/IEC 9899::1999: Programming languages — C (Second Edition), N. Y.: American National Standards Institute, 1999, c. 402-416.

4. В.Л. Безруков, А.Н. Годунов, П.Е. Назаров, В.А. Солдатов, И.И. Хоменков, Введение в ос2000, Вопросы кибернетики. Информационная безопасность. Операционные системы реального времени. Базы данных. Под. ред. чл.-корр. РАН В.Б. Бетелина, М.: Издание НИИСИ РАН, 1999, с. 76-109.

5. В.Б. Бетелин, В.А. Галатенко, А.Н. Годунов, А.И. Грюнталь, Анализ информационной безопасности систем на платформе ОС РВ БАГЕТ, Безопасность информационных технологий, № 4, 2002, с. 65–72.

6. А.И. Грюнталь, К.Г. Нархов, Методы удаленной отладки ПЛК в среде ТСАГ СПО, Труды НИИСИ РАН, том. 10, №5-6, 2020, с. 120–126.

Методы резервирования программируемых логических контроллеров семейства Багет

М.С. Аристов[1], О.В. Сердин[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, maristov@niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, serdin@niisi.ras.ru

Аннотация. В данной статье рассматриваются методы резервирования используемые в программируемых логических контроллерах семейства Багет в распределенных системах управления.

Ключевые слова: программируемые логические контроллеры, резервирование

1. Введение

Системы управления, использующие логические контроллеры в области Топливно-энергетических комплексов проектируются как безопасные системы. Для подобных систем и их компонентов необходимо использовать методы резервирования, которые удовлетворяют стандартам ГОСТ Р МЭК 61508 (1) и ГОСТ Р МЭК 61131–6 (2).

Основным методом обеспечения безопасного функционирования при возможности системных сбоев и отказов является метод горячего резервирования контроллера.

Для организации горячего резервирования и высокой степени готовности необходимо обеспечить возможность смены любых модулей системы без выключения питания и без внесения помех в работу всего комплекса.

Методы резервирования можно разделить на общие (зеркалирование) и поэлементные (дублирование процессорных модулей, модулей ввода-вывода). Выбор метода или комбинации методов резервирования осуществляется организацией-проектировщиком системы исходя из параметров, предъявляемых к системе в части безопасности, отказоустойчивости и стоимости конечного изделия.

2. Метод общего резервирования

Данный метод подразумевает полное дублирование всего комплекса управления производственным процессом. Это подразумевает создание полной копии шкафа автоматики с процессорными модулями и модулями ввода-вывода.

Шкаф автоматики «А» и его копия шкаф «Б» выполняют одну и ту же пользовательскую программу, но шкаф «Б» находится в резерве, и он не осуществляет управление объектом автоматизации. Входные и выходные сигналы до объекта автоматизации объединяются через специализированные терминальные панели.

Адреса модулей на шине связи и схемы подключения входных и выходных каналов у шкафов «А» и «Б» идентичны. Процессорные модули подключены между собой используя интерфейс Ethernet.

В процессе работы процессорные модули шкафов осуществляют обмен данными и синхронизируют между собой время и состояние прикладной программы.

В случае сбоя любого из модулей шкафа «А» приводит к его перезагрузке и потере связи с процессорным модулем шкафа «Б». Все модули вывода шкафа «А» переключают каналы на состояние выключено. Шкаф «Б» получив сигнал что связь прервалась переключается в режим основного контроллера и осуществляет управление объектом автоматизации.

У данного метода есть очевидное преимущество – это простота построение схемы управления.

Основным недостатком метода является то, что выход из строя любого элемента основного шкафа автоматики приводит к переключению на резерв всего шкафа. Таким образом при наличии в системе одного резерва и по одному отказу оборудования в основном и резервном шкафах автоматики приводит к полному отказу системы.

Для реализации данного подхода не нужны специализированные аппаратные механизмы, таким образом данный метод резервирования реализуем в программируемых логических контроллерах под управлением операционной системы реального времени семейства Багет.

3. Метод поэлементного резервирования

Метод поэлементного резервирования подразумевает что внутри одного шкафа автоматики дублируются важные элементы управления:

процессорный модуль, модули ввода-вывода и модули питания. При этом отказ одного из модулей не приводит к отказу всей системы, а программа переключается на работу с резервным модулем.

Основным преимуществом такого подхода является то, что система сохраняет работоспособность в случае нескольких отказов в разных модулях шкафа автоматики.

Основным недостатком метода является сложность проектирования системы с подобным методом резервирования. Вытекающая из этого сложность ПО может привести к дополнительным рискам с точки зрения надежности системы. Так же недостатком является сложность масштабирования этого подхода.

Встроенные системы самодиагностики и аппаратные решения в модулях программируемого логического контроллера семейства Багет позволяют реализовать данный метод резервирования.

3.1. Метод дублирования процессорного модуля

Основной процессорный модуль «А» выполняет пользовательскую программу. Резервный процессорный модуль «Б» находится в горячем резерве – исполняет ту же программу синхронизируя области данных сигналов и внутренних переменных с модулем «А». При этом обмен данными с модулями ввода-вывода осуществляется только с процессорного модуля «А». При отсутствии данных синхронизации с модуля «А», модуль «Б» переключается в основной режим и осуществляет управление объектом автоматизации.

После замены неисправного модуля «А» новый процессорный модуль осуществляет самодиагностику, синхронизирует данные с модуля «Б» и начинает работать в режиме резервного модуля.

3.2. Метод дублирование модулей ввода сигналов

Основной и резервный модули ввода сигналов располагаются на одной шине, но с разными адресами. Процессорный модуль в цикле основной программы опрашивает состояние входов обоих модулей. Для проверки целостности линий подключения каналов и результатов самодиагностики в модулях ввода предусмотрен специальные регистры, считываемые процессорным модулем.

При изменении состояния регистра самодиагностики или обрыва модуль помечается как неисправный и исключается из процедуры опроса в пользовательской программе.

3.3. Метод дублирование модулей вывода сигналов

Основной и резервный модуль располагаются на одной шине, но с разными адресами. Выходы модулей объединяются через терминальную панель. Процессорный модуль переключает состояние каналов только в одном модуле. Второй помечается как резервный и все его выходы переключаются в состояние «выключено».

В каждой итерации цикла происходит опрос регистров самодиагностики и обрыва обоих модулей: основного и резервного.

При изменении состояния регистра самодиагностики или обрыва модуль помечается как неисправный и исключается из процедуры передачи управляющих сигналов. Если основной модуль был помечен как неисправный, то программа процессорного модуля переключается на управление состояниями каналов резервного модуля.

4. Заключение

Резервирование и отказоустойчивость - два наиболее важных аспекта в системах управления производственным процессом, поскольку они решают задачи стабильности и надежности аппаратного обеспечения. В статье рассмотрено несколько методов резервирования, которые были реализованы в виде программ и библиотек применяемых в программируемых логических контроллерах под управлением операционной системы реального времени семейства Багет.

Redundancy Methods for Programmable Logic Controllers BAGET family

M. Aristov, O. Serdin

Abstract. This article discusses the methods of redundancy used in programmable logic controllers BAGET family in distributed control systems.

Keywords: programmable logic controllers, redundancy

Литература

1. IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related systems, (2010).
2. IEC 61131-6:2012 "Programmable controllers - Part 6: Functional safety", (2012).

Назначение, архитектура и функциональные возможности комплекса программ КРПО АСУ ТП

М.С. Аристов[1], А.И. Грюнталь[2], С.Г. Дышленко[3], Я.А. Зотов[4], К.Г. Нархов[5], Д.В. Яриков[6]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, maristov@niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, grntl@niisi.ras.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, dishlenko@niisi.ras.ru;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, zotov@niisi.ras.ru;

[5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, kostas@niisi.ras.ru;

[6]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, yarikov@niisi.ras.ru.

Аннотация. В статье описываются назначение, архитектура и функциональные возможности комплекса программ КРПО АСУ ТП, предназначенного для разработки и эксплуатации автоматизированных систем управления технологическими процессами.

Ключевые слова: автоматизированная система, операционная система реального времени, технологический процесс, программируемый логический контроллер, сервер ввода-вывода, программа диспетчерского управления и сбора данных.

1. Введение

В ФГУ ФНЦ НИИСИ РАН разработан комплекс программ «Комплект разработки программного обеспечения» (комплекс программ КРПО), предназначенный для разработки и эксплуатации автоматизированных систем управления технологическими процессами (АСУ ТП) широкого спектра. Целью разработки было создание отечественных программ, базирующихся на отечественной операционной системе реального времени и на отечественных средствах вычислительной техники.

Комплекс программ КРПО предназначен для разработки функционального программного обеспечения (ФПО), исполняемого на программируемых логических контроллерах (ПЛК) и непосредственно реализующего алгоритмы управления технологическими процессами и другими объектами автоматизации (ОА).

Программирование и разработка осуществлялись с учётом требований информационной безопасности и технологической независимости, необходимых для соблюдения требований стандартов ГОСТ Р 56939-2016 [1], ГОСТ Р МЭК 61508 [2] и ГОСТ Р МЭК 62443 [3].

В результате разработки КРПО часть требований информационной безопасности оказалась выполненной, другая часть требований может быть реализована на уровне разработки прикладных или общесистемных программ и/или аппаратуры. В статье [4] выполненные требования к информационной безопасности программного обеспечения подробно рассматриваются, перечисляются и указываются методы реализации других требований. Отметим, что для КРПО основу реализации требований по информационной безопасности составляет применение отечественной операционной системы и отечественной аппаратуры (микропроцессора и программируемых логических контроллеров). Это даёт возможность гарантировать отсутствие закладных элементов в программном обеспечении и в аппаратуре.

Ниже приводятся основные архитектурные решения в части программ, структурная и функциональная схема компонентов КРПО, их взаимосвязи в рамках системы, приводятся технические характеристики основных программных компонентов.

Применённые в КРПО архитектурные решения позволяют встраивать в приложения средства контролируемого выполнения, разработанные или разрабатываемые в ФГУ ФНЦ НИИСИ РАН.

2. Аппаратно-программная платформа

Комплекс программ КРПО предназначен для разработки и эксплуатации программного обес-

печения, исполняемого на программируемых логических контроллерах (ПЛК) семейства «Багет». Все ПЛК выполнены на базе отечественных микропроцессоров 1890ВМ8Я и 1890ВМ10Я [5]. Каждый контроллер включает аппаратные интерфейсы, обеспечивающие поддержку промышленных протоколов Modbus TCP [6], Modbus RTU [6], МЭК 101 [7], МЭК 104 [8], OPC UA [9], а также технологические интерфейсы UART и Ethernet.

Комплекс программ КРПО АСУ ТП предназначен для разработки и эксплуатации программ, функционирующих в среде операционной системы реального времени ОС РВ Багет ([10], [11]). Функциональное программное обеспечение (ФПО) для этой операционной системы разрабатывается с использованием технологии кросс-разработки. Собственно, разработка ФПО осуществляется на инструментальной ЭВМ (ИЭВМ), на которой установлена операционная система семейства Linux. Программа, называемая исполняемый образ, создаётся на ИЭВМ. Исполняемый образ содержит модули ФПО и необходимые для исполнения ФПО модули операционной системы. Готовый и отлаженный исполняемый образ переписывается и ПЛК. Выполнение на ПЛК исполняемого образа происходит при включении питания. Функциональное программное обеспечение, разрабатываемое для ПЛК, называется также целевой программой.

Штатные средства (без использования ИЭВМ) изменения записанных в ПЛК программ, включая ФПО и операционную систему, отсутствуют. По этой причине гарантируется целостность исполняемого образа и, как следствие, высокий уровень защищённости от кибернападений.

3. Функциональная и программная архитектура КРПО

Комплекс программ КРПО включает три основных программы: программу диспетчерского управления и сбора данных (ДУиСД), сервер ввода-вывода (СВВ) и программное обеспечение ПЛК.

Для организации информационного взаимодействия ПЛК и СВВ и СВВ и программой ДУиСД применяется проприетарный протокол MLCP (Monitoring Library Communication Protocol). Протокол MLCP является прикладной надстройкой над HTTP (HyperText Transfer Protocol) и предполагает использование в составе приложений, построенных на модели «клиент-сервер». Использование программного обеспечения, обеспечивающее поддержку данного протокола, позволяет обеспечить унифицированное окружение для приема и передачи данных, скрывающее как особенности аппаратуры, так и особенности используемой операционной системы и прочего общего программного обеспечения.

В протоколе MLCP в качестве формата представления ресурсов и данных используется JSON (JavaScript Object Notation) [12]. Единицей приёма-передачи данных является JSON-пакет, синтаксис и семантика которого определяется схемой JSON (JSON MLCP schema).

Клиентский функционал протокола MLCP реализуется посредством библиотеки, предоставляющей API для доступа к СВВ.

Управление работой ПЛК и контроль за функционированием ПЛК осуществляется с помощью программы Диспетчерского управления и сбора данных (ДУиСД), функционирующей на АРМ оператора. Программа ДУиСД включает инструментальные средства, предназначенные для создания мнемосхем, панелей и других механизмов графического управления и визуализации. Созданные программой ДУиСД в технологическом режиме элементы графического управления затем используются уже в штатном режиме для организации интерактивного графического диалога, направленного на контроль и управление программируемыми логическими контроллерами.

Взаимодействие АРМ оператора и ПЛК происходит посредством программы «Сервер ввода-вывода» (СВВ). Сервер ввода-вывода накапливает данные поступающие от одного или нескольких ПЛК. Накопленные данные периодически передаются в АРМ оператора программе ДУиСД.

Управляющие команды, создаваемые программой ДУиСД (автоматически или при участии оператора), передаются в сервер ввода-вывода, а затем из сервера ввода-вывода отсылаются в ПЛК.

4. Программное обеспечение ПЛК

Согласно штатной технологии разработки, целевая программа, исполняемая на ПЛК, пишется на алгоритмическом языке Си. Применение языка Си позволяет вести разработку ФПО, предназначенного для произвольной (в рамках аппаратных возможностей) прикладной области. Однако применение языка Си во многих случаях требует больших трудозатрат на разработку ФПО, что привело к необходимости использования специализированных инженерных языков программирования, например, МЭК 611131-3-

2016 ([13], [14]). Средства автоматизации разработки ФПО средствами этих языков в среде КРПО в настоящее время находятся в разработке и в статье не рассматриваются.

Для разработки ФПО требуются программы (драйверы), обеспечивающие работу с промышленными протоколами Modbus TCP, Modbus RTU, МЭК 101, МЭК 104, OPC UA. Информационное взаимодействие ПЛК и сервера ввода-вывода осуществляется по протоколу MLCP. Соответственно, в состав программного обеспечения ПЛК входит программа поддержки протокола MLCP.

Специальное программное обеспечение (СПО), поставляемое вместе с ПЛК, представляет собой пример функциональной программы, использующей перечисленные промышленные протоколы и программу поддержки MLCP, и может быть использовано в качестве прототипа.

Библиотеки поддержки протоколов Modbus RTU, Modbus TCP, МЭК60870-5-101, МЭК60870-5-104 и OPC UA предоставляют прикладной программный интерфейс для передачи данных по указанным протоколам. Они предназначены для создания приложений, включающих в себя обмен сообщениями с устройствами, поддерживающими данные протоколы, для целевых ЭВМ с микропроцессорами, на которых может выполняться операционная система реального времени ОС РВ Багет. Указанные библиотеки предоставляют возможность осуществлять работу приложений как в режиме сервера/slave, так и в режиме клиента/master.

5. Программа диспетчерского управления и сбора данных

Программа диспетчерского управления и сбора данных (программа ДУиСД) принадлежит к семейству SCADA-программ. Программа предназначена для автоматизации работы оператора технологического процесса и обеспечивает визуализацию состояния технологического процесса в виде мнемосхем (условное графическое отображение объекта автоматизации), графиков, окон со значениями контролируемых технологических параметров (КТП) объекта автоматизации (ОА), а также передачу команд оператора одному или нескольким программируемым логическим контроллерам (ПЛК), которые непосредственно управляют технологическим процессом. В программе ДУиСД существует журналирование отправки и получения данных, ошибок и действий оператора.

Программа ДУиСД — полностью оригинальная программа, разработанная в ФГУ ФНЦ НИИСИ РАН без использования заимствованного кода. Программа ДУиСД содержит развитые средства создания мнемосхем и позволяет создавать проекты для широкого класса задач автоматизации. Информационное взаимодействие между программой ДУиСД и ПЛК осуществляется по разработанному в ФГУ ФНЦ НИИСИ РАН проприетарному протоколу MLCP. Также возможно использование других протоколов: HTTP, Modbus TCP, TCP Socket. Программа ДУиСД функционирует под управлением операционных систем Linux и Windows. Также возможен запуск программы ДУиСД в виде тонкого клиента на системах с наличием X-сервера, в частности, на процессоре 1890ВМ8Я с операционной системой ОСРВ Багет.

Программа ДУиСД включает набор технологических средств для создания мнемосхем, средств диалога с пользователем, автоматического накопления и хранения данных о технологическом процессе. Для управления объектами автоматизации (ОА) средствами программы ДУиСД создаётся проект, представляющий собой совокупность данных, которая используется для управления конкретным ОА. Проекты хранятся в виде текстовых файлов в JSON-формате [12]. Программа ДУиСД функционирует в штатном и технологическом режимах.

В штатном режиме программа ДУиСД обеспечивает сбор, визуализацию и хранение КТП, поступающих от ОА, передачу в режиме реального времени в ПЛК команд оператора по управлению технологическим процессом, определение недопустимых отклонений значений КТП от штатных значений, оповещение о нештатных режимах работы ОА, протоколирование поступающих данных и действий оператора.

Технологический режим работы предназначен для разработки и редактирования мнемосхем ОА с помощью графического интерфейса пользователя. В технологическом режиме обеспечивается создание и редактирование иерархической структуры ОА, размещение индикаторов (примитивов, предназначенных для графической и текстовой визуализации данных, например, графиков, табло, стрелочных индикаторов) и управляющих примитивов (примитивов, используемых пользователем для передачи данных и команд в ПЛК, например, кнопок, переключателей, радиокнопок), а также настройка получаемых и отправляемых КТП.

При работе в штатном режиме происходит обмен данными между программой ДУиСД и ПЛК. Передача данных от ПЛК в ДУиСД осуществляется в виде тегов. Передача данных из ДУиСД в ПЛК осуществляется в виде тегов и команд.

Тег представляет собой пару вида <имя тега, значение>. Имя тега – это уникальный в рамках

проекта идентификатор источника данных. Значение – это действительное число. Значение тега может быть сгенерировано программным обеспечением ПЛК, либо может представлять собой конкретную выданную датчиком величину.

Команда представляет собой пару вида <номер команды, значение>.

Период обмена данными между ДУиСД и ПЛК по умолчанию составляет 1 секунду. Передача данных из ПЛК в ДУиСД происходит один раз за период обмена данными. В случае, когда полученное значение тега выходит за пределы установленного допустимого интервала, ситуация считается аварийной, и информация об этом поступает оператору.

Передача данных из ДУиСД в ПЛК также происходит один раз за период обмена данными. Для передачи в ПЛК выявляются все теги, значения которых изменились (например, в результате действий оператора) после окончания предшествующего периода обмена данными. Затем эти теги передаются в ПЛК. Аналогично устроена передача команд. За период обмена данными команды оператора накапливаются и затем передаются в ПЛК.

Передача данных от ДУиСД к ПЛК и в обратном направлении осуществляется через сервер ввода-вывода (СВВ) по протоколу MLCP. Протокол MLCP использует клиент-серверную архитектуру, где ДУиСД или ПЛК выступают в роли клиента, обращающегося к серверу за данными, а СВВ — сервера, отвечающего на запрос.

Для получения данных клиент на своей стороне создает экземпляр структуры, содержащий идентификатор ПЛК и имена тегов, которые надо получить, и затем отправляет этот экземпляр на сервер. Сервер проводит проверку, что в соответствующей таблице базы данных есть все запрашиваемые теги, возвращает в ответ структуру, содержащую имена тегов и их значения.

Для отправки данных клиент создает экземпляр структуры, содержащий идентификатор ПЛК, имена и значения тегов, которые надо отправить в СВВ, и затем отправляет созданный экземпляр структуры на сервер. Сервер проводит проверку, того что в соответствующей таблице базы данных есть все отправляемые теги и при успешном прохождении проверки записывает в базу данных новые значения тегов и возвращает признак успешного завершения действия.

В случае, если по каким-либо причинам при отправке или получении данных произошла ошибка (например, ошибка сети или попытка считать несуществующий тег), сервер в качестве ответа вернет код ошибки

6. Программа «Сервер ввода-вывода»

Программа «Сервер ввода-вывода» реализует дополнительный уровень абстракции прикладного программного интерфейса, обеспечивающий интеграцию с различными СУБД, предоставляет средства для быстрого встраивания в коммуникационные среды автоматизации технологического процесса, обеспечивает унифицированное окружение для приема и передачи данных, скрывающее как особенности аппаратуры, так и особенности используемой операционной системы и прочего общего программного обеспечения.

Программное обеспечение сервера ввода-вывода экранирует коммуникации между сервером ввода-вывода и программируемым логическим контроллером от интерфейсов системного транспорта передачи данных и предполагает использование в программах с явной многопоточностью.

С помощью программы СВВ происходит взаимодействие АРМ оператора и ПЛК. Каждый сервер ввода-вывода может быть подсоединён к нескольким ПЛК и к нескольким АРМ оператора.

В состав программы «Сервер ввода-вывода» входит библиотека программного интерфейса базы данных, которая обеспечивает разработку и отладку программного обеспечения сервера ввода-вывода совместно с различными СУБД.

Данная библиотека написана на языке программирования Си, поддерживает многопоточность и обеспечивает дополнительный уровень абстракции прикладного программного интерфейса, позволяющий эффективно интегрировать в общесистемное программное обеспечение сервера ввода-вывода различные СУБД.

Библиотека является кросс-платформенной в смысле аппаратной архитектуры, операционной системы и СУБД. В текущей реализации поддерживаются MIPS и Intel x64 аппаратные платформы, операционные системы Linux Fedora Core и ОС РВ Багет, СУБД PostgreSQL.

7. Заключение

Комплекс программ КРПО представляет собой законченную полнофункциональную систему разработки приложений для АСУ ТП.

Комплекс программ КРПО постоянно совершенствуется и модернизируется. В комплекс будут включены программы, реализующие следующие возможности:

- резервирование;
- поддержка протоколирование аварийных событий;

- ведение единого времени;
- средства включения типовых алгоритмов;
- средства программирования на языках МЭК.

Библиотека резервирования будет реализовывать резервирование центрального процессора и модулей ввода-вывода с возможностью горячей замены отдельных элементов оборудования.

Поддержка протоколирования позволит быстро проанализировать текущую ситуацию внутри системы благодаря безопасной записи событий в порядке их возникновения. Данный инструмент можно использовать для диагностики в целях минимизации простоев оборудования и анализа отключений.

Библиотека типовых алгоритмов будет содержать алгоритмы логико-программного (пошагового) управления, автоматического регулирования и алгоритмы, выполняющие логические, синтаксические, математические, аналоговодискретные преобразования сигналов. Часть алгоритмов будет связывать аппаратуру контроллера с основной массой функциональных алгоритмов.

В состав комплекса программ КРПО будут включены транслятор для преобразования программ, написанных на языках программирования, специфицированных стандартом ГОСТ Р МЭК 61131-3-2016, в программы на языке Си, среда разработки и отладки программ.

Architecture and Functionality of the KRPO Software Package

Mikhail Aristov, Andrey Gryuntal, Sergey Dyshlenko, Yaroslav Zotov, Konstantin Narkhov, Denis Yarikov

Abstract. The article describes the purpose, architecture and functionality of the KRPO software package, designed for development and operation of automated process control systems.

Keywords: automated system, real-time operating system, technological process, programmable logic controller, I/O server, supervisory control and data acquisition program.

Литература

1. ГОСТ Р 56939-2016. Защита информации. Разработка безопасного программного обеспечения. Общие требования.
2. ГОСТ Р МЭК 61508. Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью.
3. ГОСТ Р МЭК 62443. Сети промышленной коммуникации. Безопасность сетей и систем.
4. А.И. Грюнталь, С.Е. Базаева. Вопросы обеспечения кибербезопасности при разработке и использовании АСУ ТП. «Труды НИИСИ РАН», Т. 11 (2021), № 4.
5. С.И. Аряшев, П.С. Зубковский. 64-разрядные системы на кристалле с архитектурой "КОМДИВ". Наноиндустрия, №5(89), 2019 г.
6. Описание протоколов Modbus, https://modbus.org
7. ГОСТ Р МЭК 60870-5-101-2006. Описание протокола МЭК 60870-5-101.
https://docs.cntd.ru/document/1200044527
8. ГОСТ Р МЭК 60870-5-104-2004. Описание протокола МЭК 60870-5-104.
https://docs.cntd.ru/document/1200036299
9. Описание протокола OPC UA.
https://opcfoundation.org/developer-tools/specifications-unified-architecture
10. А.Н. Годунов. Операционная система реального времени Багет 3.0. Программные продукты и системы, №4, с.15-19, 2010 г.
11. А.Н. Годунов, В.А. Солдатов. Операционные системы семейства Багет (сходства, отличия и перспективы), Москва, «Программирование», №5, с.68-76, 2014 г.
12. Описание формата JSON, https://www.json.org/json-ru.html.

13. И.В. Петров. Программируемые контроллеры. Стандартные языки и приемы прикладного программирования. Москва, СОЛОН-Пресс, 2004 г.

14. ГОСТ Р МЭК 61131-3-2016 Контроллеры программируемые. Часть 3. Языки программирования.

Сложно-функциональный блок часов реального времени с доменом батарейного питания

Ю.Б. Рогаткин[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, РФ, ryb@cs.niisi.ras.ru

Аннотация. Представлен сложно-функциональный блок, в котором генерируется сигнал формы близкой к меандру на частоте внешнего кварцевого резонатора для управления часами реального времени. Блок представляет собой обособленный домен и может быть запитан, исходя из условий эксплуатации, как от внешней батарейки с номинальным напряжением +3,0В, так и от внешнего источника питания микросхемы + 3,3В, в которую блок встраивается. Сложно-функциональный блок предназначен для реализации по технологии КМОП с проектными нормами 65 нм.

Ключевые слова: часовой генератор, низкое потребление, детектирование внешнего питания

1. Введение

В последние время в России возник повышенный интерес к разработке интегральных микросхем, предназначенных для применения в системах сбора и обработки информации как в быту, так и в промышленности. Это и система «умный дом», под которой понимается комплекс решений, позволяющих автоматизировать повседневные действия, избавляя владельца от рутины. Причем это скорее не набор устройств, которыми можно «командовать» удаленно, а единая система управления ими, подразумевающая принятие решений на основе полученной информации. Это и микросхемы, которые применяются в различных электронных датчиках (давления, температуры), счетчиках электроэнергии и т.д. [1]. Одной из важных составных частей такого рода устройств является блок часов реального времени с доменом батарейного питания, который позволяет сохранить функциональную способность устройств при отключении питания и в «спящем» режимах [2].

2. Структура сложно-функционального блока

Текст второй главы, возможно, с применением *курсива*.

Данная работа посвящена задаче разработки сложно-функционального блока (СФ-блока), предназначенного для изготовления по КМОП технологии с проектными нормами 65 нм и позволяющего реализовать следующие функции:

- генерирование сигнала формы близкой к меандру на частоте внешнего кварцевого резонатора 32,678 кГц;
- работа от батарейного питания в широком диапазоне изменения напряжения последней;
- детектирование наличия внешнего напряжения питания и выработки сигнала сигнализации об его отключении.

Дополнительным требованием к блоку является его низкое потребление, а также возможность формировать дополнительное напряжение питания +1,0 В, которое может использоваться для питания цифровых доменов как внутри блока, так и вне его.

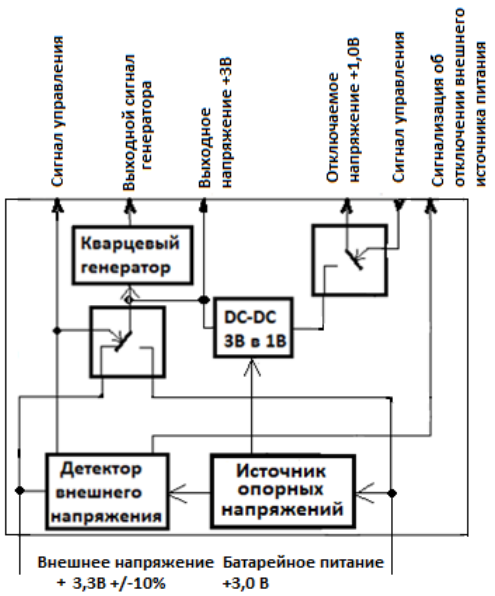

Рис. 1. Структурная схема блока часов реального времени с доменом батарейного питания.

На рисунке 1 приведена упрощенная структурная схема СФ-блока часов реального времени с доменом батарейного питания. Питание блока

осуществляется либо от внешнего источника питания +3,3 В ± 10%, либо от батарейки с напряжением + 3,0 В, например, типа CR2032. При наличии внешнего напряжения питание СФ-блока осуществляется от него, а батарейка находится в режиме энергосбережения.

Блок выдает маломощное напряжение равное напряжению внешнего источника, а также формирует маломощное стабилизированное и при необходимости отключаемое напряжение +1В для питания цифровых блоков ИМС. При пропадании внешнего напряжения (обрыве, коротком замыкании) срабатывает схема детектирования наличия внешнего напряжения, которая вырабатывает сигналы сигнализации об отключении внешнего питания в формате +3,0 В и в формате +1,0 В. В этом случае питание блока осуществляется от батарейки. На рисунке 2 приведена топология блока.

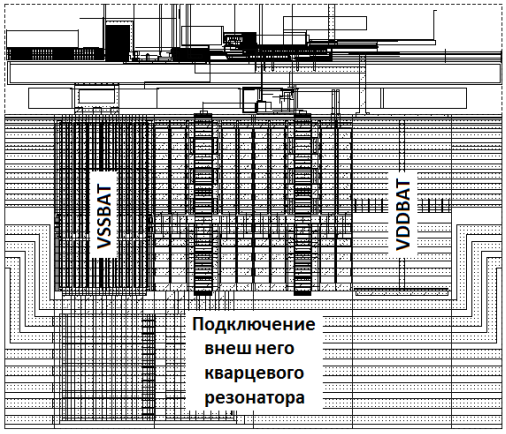

Рис. 2. Общий вид топологии СФ-блока.

Размеры блока составляют 300 мкм х 260 мкм.

3. Результаты моделирования и основные электрические характеристики

На рисунке 3 приведены результаты моделирования, иллюстрирующие работу представленного блока. При этом напряжение внешнего источника принималось равным +3,3В, а напряжение батарейки - +2,7В (возможно понижение данного напряжения до +2,2В). Исследовались следующие состояния внешнего источника питания после его включения: короткое замыкание и обрыв. В обоих случаях питания блока переключалось на питание от батарейки с сохранением работоспособности блока – наличием выходного сигнала часового генератора.

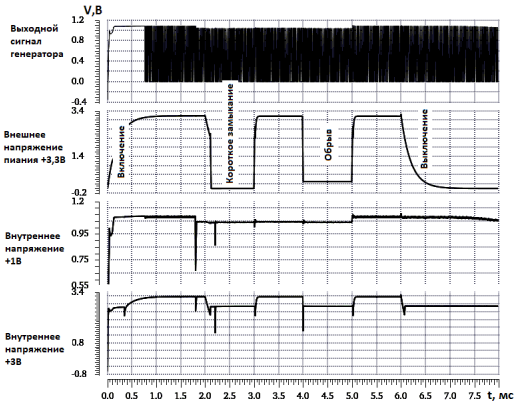

Рис. 3. Иллюстрация работы СФ-блока в различных режимах

В таблице 1 приведены основные электрические характеристики представленного СФ-блока часов реального времени.

Табл.1

Наименование параметра и размерность	Условия определения	Значение параметра
Рабочая температура t, C°	VDDBAT=3,05 В; VDD33=3.63 В;	минус 40 … плюс 85
Напряжение внешнего источника VDDB33, В	t=(-40 …+85)C°	+3,3±10%
Напряжение батарейки VDDBAT, В	t=(-40 …+85)C°	+3,05 … 2,7
Ток потребления от батарейки в случае отключения (короткого замыкания, обрыва) внешнего источника питания, мкА	t=(-40 …+85)C°; VDDBAT=3,05 В;	не более 10
Ток потребления от батарейки в	t=(-40 …+85)C°;	не более 2

случае работы от внешнего источника питания, мкА	VDDBAT=3,05 В; VDD33=3.63 В;	

4. Заключение

В результате получен блок часов реального времени, обладающий следующими отличительными характеристиками:

- широкий диапазон напряжения питания от +3,6В ло +2,7В;
- низкое потребление;
- возможность переключения источника питания при различных условиях эксплуатации, что способствует сохранения батарейного ресурса;
- наличие встроенного источника опорного напряжения.

Изложенные выше результаты были использованы при разработке микросхемы малопотребляющего микроконтроллера на основе отечественных СФ-блоков для применения в устройствах промышленного интернета вещей и разработке семейства микроконтроллеров для контроля качества электрической энергии. Публикация подготовлена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме №0580-2021-0008.

A Complex-Functional Real-Time Clock Unit with a Battery-Powered Domain

Y. B. Rogatkin

Abstract. A complex-functional block is presented in which a signal of a shape close to a meander is generated at the frequency of an external quartz resonator to control a real-time clock. The unit is a separate domain and can be powered, based on operating conditions, both from an external battery with a rated voltage of +3.0V, and from an external power source of the + 3.3V chip into which the unit is embedded. The complex-functional unit is designed for implementation using CMOS technology with design standards of 65 nm.

Keywords: clock generator, low consumption, external power detection

Литература

1. М.Павлюк, А.Коточигов, Ю.Сахно. Микроконтроллеры для счетчиков электроэнергии – современные решения. Электроника: наука, технология, бизнес, РИЦ «Техносфера» (2014), №4, 84-89.

2. Прецизионные часы реального времени: [Электронный ресурс]. URL: http://www.compel.ru/lib/54958/. (Дата обращения: 09.11.2021).

Многомерные вычеты и обобщения теоремы Кэли-Бахараша

Жгун В.С.[1,2], Бут Н.А.[2]

[1] Научно-исследовательский Институт Системных Исследований, Москва, Россия, zhgoon@mail.ru;

[2] Научно-исследовательский Университет Высшая Школа Экономики, Москва, Россия;

Аннотация. Данная работа посвящена попытке дать алгебраическое доказательство теоремы Му Лин Ли, которая является обобщением Теоремы Кэли-Бахараша на случай нерегулярного сечения. В ходе работы получилось разработать метод, с помощью которого можно построить алгебраическое доказательство, а также доказать необходимые технические результаты.

Ключевые слова: вычет, комплексное многообразие, комплекс Кошуля, спектральная последовательность, гиперкогомологии

1. Введение

Данная работа посвящена попытке дать алгебраическое доказательство основной теоремы из статьи [1] китайского математика Му Лин Ли. Доказательство теоремы в указанной статье имеет аналитическую специфику и использует структуры типа эрмитовой метрики на голоморфном расслоении, а также связности, соответствующей этой метрике, чего бы нам хотелось избежать. Отметим, что теорема Ли является обобщением а более общей теоремы, сформулированной и доказанной Ф.Гриффитсом и Дж.Харрисом из которой легко следует теорема Кэли-Бахараша о точках пересечения кривых на проективной плоскости. Заметим, что Гриффитс и Харрис дали два доказательства своей теоремы, одно содержится в монографии [3], другое - в статье [2]. Эти доказательства несколько разные по форме, но они основаны на понятии многомерного вычета и некоторых других аналитических аргументах, а также на спектральной последовательности локального функтора *Ext*, сходящаяся к глобальному функтору Ext.

Мы бы хотели получить чисто алгебраическое доказательство теоремы Ли, основываясь на спектральной последовательности гиперкогомологий. В ходе работы получилось разработать метод, на котором можно построить доказательство а также доказать необходимые технические результаты, необходимые для его алгебраизации.

2. Основная теорема

Чтобы сформулировать основную теорему введём некоторые дополнительные понятия. Пусть $s \in \Gamma(X, E)$, тогда s определяет следующий набор коротких точных последовательностей:

$$0 \longrightarrow N_i \longrightarrow E|_{Z_i} \longrightarrow V_i \longrightarrow 0 \qquad (*)$$

где N_i есть нормальное расслоение к Z_i. Данные точные последовательности получаются из следующих отображений:

$$TZ_i \hookrightarrow T\mathscr{E} \cong TX \oplus E \twoheadrightarrow N_i \hookrightarrow E$$

Также обозначим через K_X старшую внешнюю степень кокасательного расслоения к X. Теперь можно сформулировать основную теорему.

Основная теорема *Пусть X компактное аналитическое проективное многообразие размерности n, E аналитическое векторное расслоение над X ранга n. Пусть $s \in \Gamma(X, E)$, такое что множество нулей s представляется в виде конечного набора непересекающихся гладких подмногообразий Z_i в X. В предположении, что точные последовательности $(*)$ расщепляются, верно следующее утверждение. Пусть $\psi \in \Gamma(X, K_X \otimes detE)$ такое сечение, что ψ обращается в нуль на всех Z_i положительной размерности и на всех Z_i размерности 0, кроме одной, тогда ψ обращается в нуль на всех Z_i.*

Далее в работе будет удобнее рассматривать, двойственные точные последовательности

к последовательностям $(*)$. Рассмотрим отображение $E^* \longrightarrow J_i$, заданное формулой $\phi \mapsto \langle \phi, s \rangle$, и индуцированное им отображение $J_i \otimes E^* \longrightarrow J_i^2$, где J_i - это идеал Z_i. Они дают точную последовательность

$$0 \longrightarrow V_i^* \longrightarrow E^*|_{Z_i} \longrightarrow J_i / J_i^2 \longrightarrow 0,$$

где $J_i / J_i^2 = N_i^*$ конормальное расслоение. Отметим, что мы будем пытаться решать задачу в меньшей общности, а именно мы предполагаем, что среди Z_i нет дивизоров. Это не очень существенное ограничение, важное для нашего метода.

3. Пучки когомологий комплекса Кошуля

Хорошо известно, что сечение s расслоения E определяет комплекс Кошуля:

$$0 \longrightarrow \Lambda^n E^* \longrightarrow \Lambda^{n-1} E^* \longrightarrow \ldots \longrightarrow E^* \longrightarrow \mathcal{O}_X \longrightarrow 0$$

Этот комплекс получается взятием n-ой степени комплекса $E^* \xrightarrow{s^*} \mathcal{O}_X$, который является спариванием с сечением s.

Метод Ф. Гриффитса и Дж. Харриса для случая регулярного сечения s основан на применении спектральной последовательности, вычисляющей функтор $\mathrm{E}xt^\bullet(J, \Lambda^n E^*)$, где через J обозначен идеал множества $Z = \coprod_i Z_i$. Комплекс Кошуля, который мы для удобства будем обозначать буквой K, может быть проинтерпретирован как глобальный цикл на этой спектральной последовательности. Но это работает только за счёт регулярности сечения s, так как иначе K будет иметь много нетривиальных когомологий. По причинам, которые станут понятны позже, в нашем подходе роль спектральной последовательности для вычисления функтора Ext будет играть спектральная последовательность, которая вычисляет гиперкогомологии K. Чтобы вычислить вторую страницу этой спектральной последовательности, нужно сначала вычислить пучки когомологий K. Перед тем, как производить это вычисление, докажем вспомогательное утверждение.

Утверждение 1 *Пусть X неприводимое аффинное многообразие, а Z гладкое неприводимое подмногообразие коразмерности d. Пусть также набор функций $s_1, \ldots, s_d$ порождает идеал J_Z и является базисом в J_Z / J_Z^2. Тогда $s_1, \ldots, s_d$ будет регулярной последовательностью в $\mathcal{O}_X$.*

Доказательство. Гладкость Z влечёт, в частности, Коэн-Маколеевость. Для таких многообразий стандартное отображение будет

$$S^\bullet(J_Z / J_Z^2) \to \bigoplus_{k \geq 1} J_Z^k / J_Z^{k+1}$$

изоморфизмом. Это эквивалентно тому (см. [6], с. 98), что для любого линейного многочлена $p(x_1, \ldots, x_n)$ с коэффициентами в J_Z^k из того, что $p(f_1, \ldots, f_n) = 0$ следует, что $f_i \in J_Z$. Пусть теперь s_{k+1} делитель нуля в $\mathcal{O}_X / (s_1, \ldots, s_k)$. Это означает, что существует такое f и набор $f_i \in \mathcal{O}_X$, для которых $f s_{k+1} = \sum_{i=1}^{k} f_i s_i$. Следовательно $f \in J_Z$, а также все $f_i \in J_Z$. Так как f не ноль в $\mathcal{O}_X / (s_1, \ldots, s_k)$, то можно выбрать f так, что $f \in (s_d, \ldots, s_{k+1})$. После того, как мы разложим f и f_i по образующим s_j, из Коэн-Маколеевости Z можно будет сделать вывод, что коэффициенты в этом разложении также будут принадлежать J_Z. Далее можно будет перенести все мономы, содержащие s_i, где $i \leq k$, из левой части в правую. Так как f изначально не лежит в $(s_1, \ldots, s_k)$, то после переноса, левая часть не будет равна нулю. Тем самым, действуя по индукции, мы получим, что $f \in \widetilde{J} = \bigcap_i J_Z^i$ по модулю $(s_1, \ldots, s_k)$. Локализуя $\widetilde{J}$ по простому идеалу J_Z, по лемме Накаямы мы получим, что $\widetilde{J} = 0$ в локальном кольце, так как

$J_Z \widetilde{J} \subset \widetilde{J}$. Так как O_X вкладывается в локальное кольцо для идеала J_Z, то и в кольце O_X будет выполнено $\widetilde{J} = 0$. Следовательно, $f = 0$ по модулю $(s_1,...,s_k)$, а значит s_{k+1} не делитель нуля в $\mathcal{O}_X / (s_1,\ldots,s_k)$.

Теперь мы можем перейти к вычислению когомологий комплекса Кошуля. Выберем градуировку так, что $\mathcal{O}_X$ находится в n-ой градуировке, а $\Lambda^n E^*$ в нулевой. Тогда верно следующее утверждение.

Утверждение 2 .

$$\mathcal{H}^{n-j}(K) = \bigoplus_i \Lambda^j V_i^*$$

Доказательство. Заметим, что вне нулей сечения s комплекс Кошуля будет точным. Это значит, что его пучки когомологий имеют носитель в Z и следовательно распадаются в прямую сумму пучков с носителями в Z_i. Поэтому считать пучки когомологий можно отдельно вдоль каждой компоненты. Пусть U некоторая аффинная окрестность Z_i, в которой можно выбрать такую тривиализацию E^*, что $E^*|_U = \widetilde{V}_i^* \oplus \widetilde{N_i}^*$, где $\widetilde{V}_i^*|_{Z_i} = V_i^*$ и $\widetilde{N_i}^*|_{Z_i} = N_i^*$. Выберем базис $e_1,\ldots e_{d_i}$ в $\widetilde{V}_i^*$ и $f_1,\ldots f_{k_i}$ в $\widetilde{N_i}^*$, где $d_i = \dim V_i^*, d_i + k_i = n$. Из того, как был выбран базис, следует, что он будет согласован с короткими точными последовательностями в условии основной теоремы, то есть $\{\langle f_l, s\rangle \bmod J_i^2\}_{l=1,\ldots,k_i}$ образуют базис в N_i^*, а $\langle e_m, s\rangle \in J_i^2 \ \forall m = 1,\ldots,d_i$. В окрестности U комплекс K можно рассматривать как комплекс Кошуля кольца $\mathcal{O}_U$ для последовательности

$$\left(\langle f_1, s\rangle,\ldots,\langle f_{k_i}, s\rangle, \langle e_1, s\rangle,\ldots,\langle e_{d_i}, s\rangle\right).$$

Далее мы воспользуемся двумя фактами. Во-первых, из утверждения 1 следует, что последовательность $\left(\langle f_1, s\rangle,\ldots,\langle f_{k_i}, s\rangle\right)$ будет регулярной в $\mathcal{O}_U$. Во-вторых, так как $\langle f_l, s\rangle$ порождают идеал J_i, а $\langle e_m, s\rangle \in J_i^2 \subset J_i$, то комплекс Кошуля для исходной последовательности будет изоморфен комплексу Кошуля для последовательности $\left(\langle f_1, s\rangle,\ldots,\langle f_{k_i}, s\rangle, 0,\ldots,0\right)$ (см. [5], с. 429), а сам изоморфизм осуществляется следующей

$$\widetilde{A} = \begin{pmatrix} E & 0 \\ -A & E \end{pmatrix}$$

блочной матрицей:

Здесь A является матрицей коэффициентов разложения $\langle e_m, s\rangle = \Sigma_{l=1}^{k_i} a_{ml} \langle f_l, s\rangle$, где $a_{ml} \in J_i$, так как $\langle e_m, s\rangle \in J_i^2$, поэтому $A|_{Z_i} = 0$ и $\widetilde{A}|_{Z_i} = Id$. Этот явный вид изоморфизма и позволяет явно вычислить пучки когомологий вдоль компоненты Z_i. Посмотрим, как $\widetilde{A}$ действует на разложение $E^*|_U$. $A\left(\widetilde{V}_i^* \oplus \widetilde{N_i}^*\right) = W_i^* \oplus \widetilde{N_i}^*$, где W_i получается применением $\widetilde{A}$ к $\widetilde{V}_i^*$ и обладает следующими свойствами, которые непосредственно следуют из свойств матрицы $\widetilde{A}$: во-первых, $W_i^*|_{Z_i} = V_i^*$, а во-вторых, $\langle W_i^*, s\rangle = 0$. Теперь вычислим пучки когомологий K. Из утверждения [5] следует, что локально K представим как тензорное произведение комплекса Кошуля для регулярной последовательности $\left(\langle f_1, s\rangle,\ldots,\langle f_{k_i}, s\rangle\right)$ и комплекса Кошуля для последовательности из нулей длины d_i, поэтому локально у такого комплекса не будет когомологий в градуировке меньшей, чем k_i, поэтому мы заранее можем положить при вычислении $\mathcal{H}^{n-j}(K)|_U, j \le d_i$. Рассмотрим для выбранного j следующую часть комплекса Кошуля:

$$\Lambda^{j+1} E^*|_U \longrightarrow \Lambda^j E^*|_U \longrightarrow \Lambda^{j-1} E^*|_U$$

Теперь, пользуясь $E^*|_U = W_i^* \oplus \widetilde{N_i}^*$, разложим дифференциал в этой части в прямую

$$\begin{array}{ccccc}
\Lambda^{j+1}W_i^* & \longrightarrow & 0 & \longrightarrow & 0 \\
\oplus & & \oplus & & \oplus \\
\Lambda^j W_i^* \otimes \widetilde{N_i}^* & \longrightarrow & \Lambda^j W_i^* & \longrightarrow & 0 \\
\oplus & & \oplus & & \oplus \\
\Lambda^{j-1}W_i^* \otimes \Lambda^2\widetilde{N_i}^* & \longrightarrow & \Lambda^{j-1}W_i^* \otimes \widetilde{N_i}^* & \longrightarrow & \Lambda^{j-1}W_i^* \\
\oplus & & \oplus & & \oplus \\
\vdots & & \vdots & & \vdots \\
\oplus & & \oplus & & \oplus \\
\Lambda^{j+1}\widetilde{N_i}^* & \longrightarrow & \Lambda^j\widetilde{N_i}^* & \longrightarrow & \Lambda^{j-1}\widetilde{N_i}^*
\end{array}$$

сумму отображений:

Заметим, что для всех строчек, кроме первой и второй, прямое слагаемое дифференциала представляет собой точную в центральном члене последовательность, тензорно умноженную на векторное расслоение, так как конормальное расслоение определяет комплекс Кошуля для регулярной последовательности, а на $\Lambda^p W_i^*$ дифференциал действует нулём для всех p. Следовательно пучки когомологий могут

$$\Lambda^j W_i^* \otimes \widetilde{N_i}^* \longrightarrow \Lambda^j W_i^* \longrightarrow 0$$

приходить только из второй строчки: Ядром тут будет $\Lambda^j W_i^*$, а образом $\Lambda^j W_i^* \otimes J_i$, так как $\langle \Lambda^j W_i^* \otimes \widetilde{N_i}^*, s\rangle = \Lambda^j W_i^* \otimes \langle \widetilde{N_i}^*, s\rangle = \Lambda^j W_i^* \otimes J_i$ Следовательно когомологии вдоль Z_i будут равны $\Lambda^j W_i^* \otimes \mathcal{O}_{Z_i} = \Lambda^j W_i^*|_{Z_i} = \Lambda^j V_i^*$. Теперь остаётся только взять прямую сумму по всем компонентам и мы получим то, что требовалось доказать.

Среди всех компонент для данной градуировки пучков когомологий комплекса Кошуля есть некоторые выделенные, а именно такие Z_i, что градуировка равна $n - d_i$.

Напомним формулировку двойственности Серра.

Утверждение 3 *(Двойственность Серра) Пусть X компактное проективное комплексное многообразие размерности n и $\mathcal{F}$ когерентный пучок на X. Тогда* $H^i(X;\mathcal{F})^* = \mathrm{E}xt^{n-i}(\mathcal{F}, K_X)$. *Если $\mathcal{F}$ локально свободный пучок, то* $H^i(X;\mathcal{F})^* = H^{n-i}(X;\mathcal{F}^* \otimes K_X)$.

Используя двойственность Серра, мы можем произвести следующее вычисление.

Утверждение 4

$$H^0(Z_i; detE \otimes K_X)^* = H^{d_i}(Z_i; \Lambda^{d_i} V_i^*)$$

Доказательство. По двойственности Серра

$$H^0(Z_i; detE \otimes K_X)^* = H^{d_i}(Z_i; (detE \otimes K_X)^* \otimes K_{Z_i}).$$

для $Z_{i,}$

Теперь, используя формулы

$$K_X|_{Z_i} = K_{Z_i} \otimes det(N_{Z_i}^*)$$

$$detE^*|_{Z_i} = det(V_i^*) \otimes det(N_{Z_i}^*),$$

получим,

$$\begin{gathered}(detE \otimes K_X)^* \otimes K_{Z_i} = \\ = det(V_i^*) \otimes det(N_i^*) \otimes det(N_i) \otimes K_{Z_i}^* \otimes K_{Z_i} = \\ = det(V_i^*) = \Lambda^{d_i} V_i^*.\end{gathered}$$

Это вычисление завершает доказательство.

4. Метод Ф. Гриффитса и Дж. Харриса

Ранее уже упоминалось, что существует несколько алгебраических доказательств более слабой версии основной теоремы, где на $s \in \Gamma(X, E)$ накладывается условие регулярности. Одно из них было дано Ф.Гриффитсом и Дж. Харрисом (см. [3], с. 774-776). Именно оно будет нас интересовать. Поскольку в монографии, на которую мы ссылаемся, это доказательство дано эскизно, здесь оно будет представлено с некоторыми важными комментариями и снабжено одним дополнительным вычислением. В ходе доказательства мы попытаемся сделать акцент на тех моментах, которые пропущены в оригинальном доказательстве, и на то, каким образом это доказательство может быть расширено до доказательства основной теоремы.

Для начала мы докажем вспомогательное утверждение про функтор $\mathcal{E}xt^\bullet(J; \Lambda^n E^*)$.

Утверждение 5 *Пусть* $s \in \Gamma(X,E)$ *регулярное сечение с множеством нулей* Z *,идеал которого обозначим за* J *. Тогда*

$$\mathcal{E}xt^0(J,\Lambda^n E^*) = \Lambda^n E^*,\ \mathcal{E}xt^{n-1}(J,\Lambda^n E^*) = \mathcal{O}_Z,$$
$$\mathcal{E}xt^j(J,\Lambda^n E^*) = 0$$

если $j \neq 0, n-1$.

Доказатеоьство. Сначала покажем, что $\mathcal{H}om(\Lambda^j E^*, \Lambda^n E^*) = \Lambda^{n-j} E^*$. Рассмотрим стандартное отображение $\Lambda^j E^* \otimes \Lambda^{n-j} E^* \longrightarrow \Lambda^n E^*$, которое локально является невырожденным спариванием. Если тензорно умножить это спаривание на $\Lambda^n E$, то получится невырожденное спаривание на уровне пучков . $\Lambda^j E^* \otimes \left(\Lambda^{n-j} E^* \otimes \Lambda^n E\right) \longrightarrow \mathcal{O}_X$

Поэтому

$$\mathcal{H}om(\Lambda^j E^*, \mathcal{O}_X) = \Lambda^{n-j} E^* \otimes \Lambda^n E.$$

Далее остаётся заметить, что, так как $\Lambda^n E^*$ локально свободный пучок,

$$\mathcal{H}om(\Lambda^j E^*, \Lambda^n E^*) = \mathcal{H}om(\Lambda^j E^*, \mathcal{O}_X) \otimes \Lambda^n E^*.$$

Для вычисления функтора E : рассмотрим комплекс Кошуля в качестве резольвенты J. Возьмём кусок K :

$$0 \longrightarrow \Lambda^n E^* \longrightarrow \Lambda^{n-1} E^* \longrightarrow \ldots \longrightarrow \Lambda^2 E^* \longrightarrow E^* \longrightarrow 0$$

Применим к нему функтор $\mathcal{H}om(\_\_, \Lambda^n E^*)$ и получим другой кусок K .

$$0 \longrightarrow \Lambda^{n-1} E^* \longrightarrow \Lambda^{n-2} E^* \longrightarrow \ldots \longrightarrow E^* \longrightarrow \mathcal{O}_X \longrightarrow 0$$

Убедиться в том, что дифференциал будет как у комплекса Кошуля легко, расписав всё в локальных координатах. Этот факт является обобщением самодвойственности комплекса Кошуля для последовательности элементов кольца на случай сечения векторного расслоения. Когомологии получившегося комплекса, по определению, являются пучками $\mathcal{E}xt^\bullet(J,\Lambda^n E^*)$. .

Поскольку изначальный комплекс Кошуля имеет когомологии только в старшей градуировке и они равны $\mathcal{O}_Z$, имеем $\mathcal{E}xt^{n-1}(J,\Lambda^n E^*) = \mathcal{O}_Z$ и $\mathcal{E}xt^j(J,\Lambda^n E^*) = 0$, если $j \neq 0, n-1$.

Чтобы найти $\mathcal{E}xt^0(J;\Lambda^n E^*)$ заметим, что . $\mathcal{E}xt^0(J;\Lambda^n E^*) = \ker(\Lambda^{n-1} E^* \to \Lambda^{n-2} E^*)$. Так как мы предположили, что среди нулей s нет дивизоров, у комплекса Кошуля, по утверждению 2, нет первых когомологий. Тогда имеем точность следующего куска из начала комплекса Кошуля:

$$0 \longrightarrow \Lambda^n E^* \longrightarrow \Lambda^{n-1} E^* \longrightarrow \Lambda^{n-2} E^*$$

Из точности делаем вывод, что $\ker(\Lambda^{n-1} E^* \longrightarrow \Lambda^{n-2} E^*) = Im(\Lambda^n E^* \longrightarrow \Lambda^{n-1} E^*) = \Lambda^n E^*$. Это завершает доказательство утверждения.

Следующее утверждение относится к функторам Ext.

Теперь можем провести доказательство следующей теоремы.

Теорема. *Пусть* X *компактное аналитическое проективное многообразие размерности* n, E *аналитическое векторное расслоение над* X *ранга* n. *Пусть* $s \in \Gamma(X,E)$ *регулярное сечение с множеством нулей* $Z = \coprod_i Z_i$. *Тогда, если сечение* $\psi \in \Gamma(X; det(E) \otimes K_X)$ *обращается в нуль на всех* Z_i*, кроме одной, то* ψ *обращается в нуль на всех* Z_i.

Доказательство. Рассмотрим спектральную последовательность, которая вычисляет функтор Ext. Её вторая страница имеет вид $E_2^{pq} = H^p(X; \mathcal{E}xt^q(J,\Lambda^n E^*))$.. Рассмотрим элемент $e \in \mathrm{E}xt^{n-1}(J,\Lambda^n E^*)$, который соответствует комплексу Кошуля. Когомологически он представляется элементом $\mathrm{I}d \in Hom(\Lambda^n E^*, \Lambda^n E^*)$ или что эквивалентно $1_X \in H^0(X;\mathcal{O}_X)$. Индуцированные ло-

$$1_{Z_i} \in H^0(Z_i;\mathcal{O}_{Z_i}) = H^0(Z_i;\mathcal{E}xt^{n-1}(J;\Lambda^n E^*)).$$

кальные классы на Z_i будут иметь вид

Следовательно элемент $\sum_i 1_{Z_i} \in H^0(X;\mathcal{E}xt^{n-1}(J,\Lambda^n E^*))$ представляет глобальный цикл. Так как в данном случае из утверждения 5 следует, что спектральная последовательность имеет сферичный слой, то есть ненулевые элементы стоят только в первой и $n-1$-ой строке, то дифференциал с n-ой страницы действует между группами со второй страницы $d_n : E_2^{0,n-1} \longrightarrow E_2^{n,0}$, который представляет из себя отображение $d_n : H^0(X;\mathcal{E}xt^{n-1}(J,\Lambda^n E^*)) \to H^n(X;\Lambda^n E^*)$. Следовательно элемент $\sum_i d_n(1_{Z_i}) = 0$. Далее в доказательстве теоремы нам придётся существенно отойти от доказательства, предложенного у Ф. Гриффитса и Дж. Харриса. В их доказательстве спаривание функционала $d_n(1_{Z_i})$ с сечением ψ интерпретируется как многомерный вычет в изолированном нуле сечения векторного расслоения $res_{Z_i}(\psi s)$ и из этого получается утверждение теоремы (подробнее про многомерные вычеты для точки и про их свойства можно посмотреть также у Ф. Гриффитса и Дж. Харриса (см. [3], гл. 5)).

Чтобы алгебраизовать доказательство из [3] и чтобы функционалы $d_n(1_{Z_i})$ действительно можно было бы интерпретировать как вычеты, необходимо показать, что они обладают следующими тремя свойствами: $\sum_i d_n(1_{Z_i}) = 0$, $\langle d_n(1_{Z_i}),\psi\rangle = 0$ при $\psi|_{Z_i} = 0$, $\langle d_n(1_{Z_i}),\psi\rangle \neq 0$ при $\psi|_{Z_i} \neq 0$. Из утверждения 6 и того факта, что $E_2^{0,n-1}$ и $E_2^{n,0}$ стабилизируются после действия d_n, следует, что $\ker(d_n)$ порождено $\sum_i 1_{Z_i}$, так как $\mathrm{E}xt^{n-1}(J,\Lambda^n E^*) = \mathbb{C}$, а $Im(d_n) = H^n(X;\Lambda^n E^*)$, так как $\mathrm{E}xt^n(J,\Lambda^n E^*) = 0$. Из вышесказанного следует, что $d_n(1_{Z_i})$ есть набор образующих в $H^n(X;\Lambda^n E^*)$ с единственным соотношением $\sum_i d_n(1_{Z_i}) = 0$. Поэтому из всех требуемых свойств к этому набору функционалов осталось проверить только второе, то есть $\langle d_n(1_{Z_i}),\psi\rangle = 0$ при $\psi|_{Z_i} = 0$. Для этого нужно будет написать явные локальные формулы для когомологических классов в $H^n(X;\Lambda^n E^*)$, которые представляют рассматриваемые функционалы. Для удобства выделим вычисление локальной формулы для этого функционала в отдельное утверждение:

Утверждение 7 *Если* $\psi|_{Z_i} = 0$, *то* $\langle d_n(1_{Z_i}),\psi\rangle = 0$.

В доказательстве этой теоремы мы использовали два существенных соображения: первое состоит в том, что мы используем глобальный цикл на спектральной последовательности, а второе, что на второй странице глобальный элемент распадается в сумму локальных по компонентам. Эти моменты мы сохраним в конструкции, которую сейчас опишем.

5. Гиперкогомологии комплекса Кошуля

В этом разделе мы попытаемся предложить путь доказательства основной теоремы, который будет обобщать идеи доказательства, разобранного в предыдущем разделе.

Отправной точкой этого подхода будет идея использовать не спектральную последовательность для вычисления глобального функтора Ext через локальный, а спектральную последовательность гиперкогомологий комплекса Кошуля. Эта идея связана с тем, что в нашем случае комплекс Кошуля не является резольвентой, поэтому не даёт глобального цикла на спектральной последовательности для Ext, а также у нас не получится вычислять функторы $\mathcal{E}xt$, используя комплекс Кошуля. Более того пучки гиперкогомологий неточного комплекса Кошуля будут иметь носитель в множетве нулей сечения. Более того поскольку пучок может быть разложен в сумму пучков, сосредоточенных в непересекающихся компонентах носителя, это даст аналог утверждения об аддитивности вычета. А именно того факта, что глобальный вычет раскладывается в сумму своих локальных значений, вычисленных в окрестности компонент связности. Тот факт, что мы можем вычислить пучки когомологий комплекса Кошуля, позволяет получить информацию о гиперкогомологиях. Для начала обозначим через K' обрезанный комплекс Кошуля:

$$0 \longrightarrow \Lambda^{n-1}E^* \longrightarrow \Lambda^{n-2}E^* \longrightarrow \ldots \longrightarrow E^* \longrightarrow \mathcal{O}_X \longrightarrow 0$$

Градуировка здесь возрастает вдоль дифференциала и $\Lambda^{n-1}E^*$ находится в нулевой градуировке. Теперь, чтобы вычислить гипрекогомологии, нужно написать резольвенту для K' и применить к ней функтор глобальных сечений. Получится бикомплекс, который мы будем обозначать L. Его когомологии и будут гиперкогомологиями K'. В качестве резольвенты будет удобно брать резольвенту Чеха.

Найдём вторую страницу спектральной последовательности для гиперкогомологий K'. Как известно, $E_2^{pq} = H^p(X; \mathcal{H}^q(K'))$. Пучки когомологий K' такие же как у $K[-1]$, кроме градуировки 0. В доказательстве утверждения 5 мы уже выяснили, что если среди Z_i нет дивизоров, то $\mathcal{H}^0(K') = \Lambda^n E^*$. Теперь понятно, что $E_2^{j,n-j-1} = \bigoplus_i H^j(Z_i; \Lambda^j V_i^*)$, а $E_2^{n,0} = H^n(X; \Lambda^n E^*)$. Это уже очень напоминает то, что было у Ф. Гриффитса и Дж. Харриса. Локальные классы, распределённые по $n-1$-ой диагонали спектральной последовательности, дифференциалами собираются в $H^n(X; \Lambda^n E^*)$. Далее мы более строго поясним, что это значит, а также покажем, что нас интересуют не все классы на диагонали.

Утверждение 8. *Пусть* $\omega \in H^{d_i}(Z_i; \Lambda^{d_i} V_i^*) \subset E_2^{d_i, n-d_i-1}$. *Тогда* $\omega' \in \bigoplus_{p+q=n-1} L^{pq}$ *существует такой* $\omega' \in Z_{n-d_i}^{d_i, n-d_i-1}$, *представляющий* ω, *что* . *Здесь использованы стандартные обозначения из теории спектральных последовательностей, в которых* Z_k^{pq} *обозначает такие элементы бикомплекса бистепени* (p,q), *что дифференциал отображается на* k *-тый член стандартной фильтрации на бикомплексе глубже, чем находятся элементы бистепени* (p,q).

Доказательство. Опишем явно бикомплекс L. Зафиксируем открытое покрытие $\mathcal{U}$, состоящее из таких аффинных множеств, что каждое открытое множество из $\mathcal{U}$ пересекается с не более чем одной компонентой Z_i. Тогда $L^{p,q} = \check{C}^p(\mathcal{U}; K'^q)$. Как и раньше, в бикомплексе действуют два дифференциала: дифференциал Чеха d_c и дифференциал Кошуля d_k.

Приступим к доказательству утверждения. Пусть $\omega^1 \in \check{C}^{d_i}(\mathcal{U}, \Lambda^{d_i} E^*)$ представляет ω и обладает тем свойством, что ω^1 отлично от нуля только в окрестностях, содержащих Z_i. Так как ω^1 представляет элемент из E_2, существует $\omega^2 \in C^{d_i+1}(\mathcal{U}, \Lambda^{d_i+1} E^*)$ с тем же свойством и такое, что $d_c\omega^1 + d_k\omega^2 = 0$. Рассмотрим $d_c\omega^2$. Ясно, что $d_k(d_c\omega^2) = d_c(d_k\omega^2) = -d_c^2\omega^1 = 0$. Получается, что $d_c\omega^2$ является циклом относительно дифференциала Кошуля. Но так как в окрестности компоненты Z_i и в градуировке меньшей чем $n - d_i - 1$ комплекс K' когомологий не имеет, то этот цикл должен быть границей, то есть должен существовать элемент $\omega^3 \in \check{C}^{d_i+2}(\mathcal{U}, \Lambda^{d_i+2} E^*)$ ненулевой только в окрестностях, пересекающихся с Z_i, и такой, что $d_c\omega^2 + d_k\omega^3 = 0$. Действуя по индукции, построим такой набор элементов $\omega^1, \omega^2, \ldots, \omega^{n-d_i}$, где $\omega^j \in \check{C}^{d_i+j-1}(\mathcal{U}, \Lambda^{d_i+j-1} E^*)$, что $d_c\omega^j + d_k\omega^{j+1} = 0$. Тогда, если положить $\omega' = \omega^1 + \ldots + \omega^{n-d_i}$, то ω' очевидно представляет ω, а также $d_L\omega' = (d_c + d_k)(\omega^1 + \ldots + \omega^{n-d_i}) = d_c\omega^{n-d_i} \in \check{C}^n(\mathcal{U}, \Lambda^{n-1} E^*)$, а это и требовалось доказать.

Заметим, что полученный в доказательстве элемент $d_c\omega^{n-d_i}$ является циклом относительно обоих дифференциалов, поэтому он даёт некоторый элемент в $H^n(X; \Lambda^n E^*)$. Обозначим его $r(\omega)$. Он определён неоднозначно и степень его неоднозначности зависит от образов дифференциалов $d_2, \ldots, d_{n-d_i-1}$ на спектральной по-

следовательности. Есть шанс, что эта неоднозначность ничему не мешает, так как все эти дифференциалы дают элементы в окрестности компонент размерности большей чем d_i, но ответ на этот вопрос в данной работе найти не удалось. Важно, что если набор $\omega_i \in H^{d_i}(Z_i;\Lambda^{d_i}V_i^*)$ представляет некоторый глобальный цикл, то можно подобрать такие $r(\omega_i)$, что $\sum_i r(\omega_i)=0$. Это следует из того, что глобальный цикл должен быть циклом на бикомплексе. Такая формула является аналогом факта, что сумма вычетов по всем полюсам равна нулю, где вычет определяется с помощью r. Чтобы эта конструкция дала нам доказательство основной теоремы и алгебраическую интерпретацию вычета, нужно также доказать, что сечение расслоения $K_X \otimes detE$, которое обращается в нуль на компоненте Z_i, будет спариваться нулём с теми функционалами, которые предлагается интерпретировать как вычеты.

6. Заключение

В данной работе получилось найти абстрактный способ вычислять вычеты вдоль компонент положительной размерности нулей сечения s. То, что этот вычет включён в спектральную последовательность гиперкогомологий комплекса Кошуля, позволяет написать глобальную формулу, в которой сумма локальных вычетов по компонентам будет давать нулевой функционал на $\Gamma(X;K_X\otimes\Lambda^n E^*)$. Это первый этап в доказательстве основной теоремы.

Чтобы завершить доказательство предложенным способом, необходимо сделать две вещи. Во-первых, нужно написать более явную локальную формулу для вычета и изучить с её помощью его свойства. Во-вторых, необходимо найти глобальный класс гиперкогомологий комплекса Кошуля, который будет правильным образом распределяться на диагонали второй страницы спектральной последовательности. Это позволит написать глобальную формулу, обобщающую тот факт, что сумма вычетов по дискретному множеству нулей регулярного сечения векторного расслоения равна нулю. Эта формула является ключевой в доказательстве обобщения теоремы Кэли-Бахараша.

ИСТОЧНИК ФИНАНСИРОВАНИЯ: Работа Жгуна В.С. была выполнена в рамках государственного задания по проведению фундаментальных научных исследований по проекту № 0580-2021-0011.

On Higher Dimensional Residues and the Generalisations of Cayley-Bacharash Theorem

Zhgoon V.S., But N.A.

Abstract. This paper is dedicated to an attempt to give an algebraic proof of the Mu Lin Li theorem, which is a generalisation of the Cayley-Bacharach Theorem on the case of an irregular section. We developed a method on which the proof can be based, also we managed to prove some necessary technical results for the algebraisation.

Keywords: residue, complex manifold, Koszul complex, spectral sequence, hypercohomology.

Литература

1. Mu-Lin Li, Virtual residue and generalized Cayley-Bacharach Theorem. In:arXiv:1801.00534v2.
2. Phillip Griffiths and Joseph Harris, Residues and zero-cycles on algebraic varieties, Ann. Math. **108** (1978), 461-505.
3. Гриффитс Ф., Харрис Дж., Принципы алгебраической геометрии: Пер. с англ. - М.: Мир, 1982. - Т. 2 - 366 с.
4. Хартсхорн Р., Алгебраическая геометрия: Пер. с англ. - М.: Мир, 1981. - 597 с.
5. David Eisenbud, Commutative Algebra with a View Toward Algebraic Geometry, (New York: Springer, 1995) - 800 p.
6. Hideyuki Matsumura, Commutative Algebra (Benjamin/Cummings Publishing Company, 1980) - 313 p.

Механизм интерактивного взаимодействия преподавателя со студентами в цифровой образовательной платформе Мирера

А.Г. Леонов[1], К.А. Мащенко[2], А.Е. Орловский[3], А.В. Шляхов[4]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, МГУ им. М.В.Ломоносова, Москва, Россия, МПГУ, Москва, Россия, Государственный университет управления, Москва, Россия, dr.l@vip.niisi.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, kirill010399@vip.niisi.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, orlovskiy@vip.niisi.ru;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, Shlyxovav@mail.ru

Аннотация. Создание цифровых образовательных платформ и сред предоставляет педагогам возможность интенсифицировать образовательный процесс, упрощая и расширяя способы внеаудиторной коммуникации со студентами. С другой стороны, студенту предоставляется возможность осваивать изучаемые компетенции, отправлять задания для автоматической проверки, в удобной форме в любое время, при наличии связи с цифровой образовательной платформой. Разработчики платформы в свою очередь должны предоставить удобный и надежный способ поддержки такого взаимодействия. В статье излагается вариант решения этой проблемы на примере авторской цифровой образовательной платформы Мирера. Отмечается возможность использования подобного механизма для реализации альтернативного подхода к прокторингу.

Ключевые слова: цифровая образовательная платформа, цифровая образовательная среда Мирера, веб-сокеты, черновики

1. Введение

Цифровая трансформация образовательных процессов подразумевает не только создание цифровых образовательных курсов или формирование гипертекстовых учебных материалов. К этому процессу, без сомнения, также можно отнести новые цифровые формы взаимодействия преподавателя со студентами во время прохождения курса. Современные технологии обучения формируют особые требования к педагогам, когда взаимодействие со слушателями не ограничивается только временным пространством аудиторной работы или онлайн консультациями [1]. Вооруженные современными цифровыми каналами связи, включая интерактив, предоставляемый чатами социальных сетей и мессенджеров, педагог включен в непрерываемый процесс взаимодействия со студентами. Так, во время выполнения заданий по программированию или по другим, в том числе, естественно-научным предметам для преподавателя важен не только финальный результат, который, например, является законченной программой на одном из языков программирования или текстовым ответом на заданную тему, но и сам процесс выполнения задания: этапы отладки программы или написание текстового ответа [2]. Работа студента над заданием предполагает подготовку ответа, исправление, добавление и прочие изменения текста, кода, которые могут происходить в совершенно произвольном порядке в различные моменты времени [3]. При этом, поскольку время сдачи студентом выполненного задания ограничено лишь временными рамками и расписанием курса, то интервалы между сеансами работы над ответом у студентов могут достигать по продолжительности нескольких дней, недель. С другой стороны, с точки зрения цифровой образовательной среды, работа на заданием - это некоторые элементарные операции работы с текстом (не важно, речь идет о математическом тексте, программе или просто текстовом ответе: добавление, изменение или удаление символов, слов, строк в редакторах цифровой образовательной среды, в надежде, что дистанционная работа допускает сохранение введенной информации, вне зависимости от надежности соединения компьютера/смартфона студента с минимизацией потери введенной информации в случае потери связи с сервером [4].

Разнообразие возможных сценариев набора текста пользователем требует создания механизма, который позволяет сохранять текущий результат ввода, не нагружая ни серверную часть интерфейса образовательной системы, ни клиентскую часть пользователя, который также должен предоставить возможность сохранять текущую копию в качестве результата выполняемого

задания. Такой подход позволит преподавателю видеть черновики студента и при необходимости прийти ему на помощь и проконсультировать в режиме онлайн. Этот механизм встроен в цифровую образовательную платформу (ЦОП) Мирера и использует интегрированные редакторы, использующие технологию веб-сокетов [5][6].

2. Механизм попыток

В ЦОП Мирера курсы состоят из тем, темы делятся на группы зачетных заданий: контесты, контрольные, тренировочные и подготовительные тесты. При этом студенту часто предоставляется возможность отправлять на автоматизированную проверку задание неограниченное число раз. Каждый такой акт сдачи решения задания в ЦОП Мирера называют «попыткой». Информация о попытке, привязаннная к пользователю, характеризуется статусом, хранит само решение, а также метаинформацию для внутреннего функционирования среды.

Попытка создается в тот момент, когда пользователь готов сдать решение и отправляет его на проверку (кнопка «Проверка решения»). При этом формируется информация об авторе попытки, задаче, привязанной к ней и способе проверки, а затем в зависимости от этого способа проверки, попытка либо отправляется в тестирующую систему, где получает результат и статус тестирования, либо, в случае ручного тестирования, отправляется преподавателю для ее проверки.

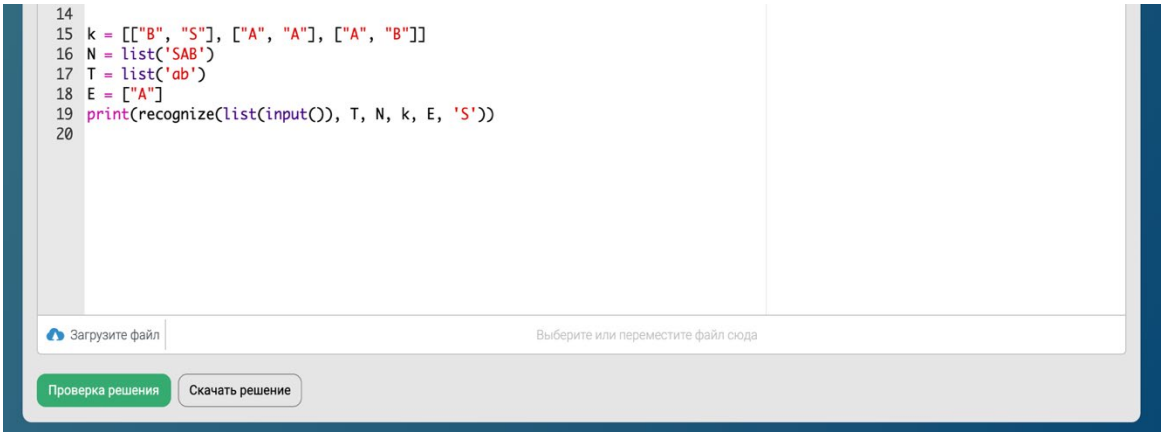

Рис. 1 Интерфейс проверки решения

Также, при соответствующих настройках, попытка может дополнительно отправиться на внешние сервисы, такие как проверка на плагиат, сравнение изображений и т.п. [7].

На основании попыток и их статусов вычисляется количество баллов, которое студент сможет набрать за решение задачи. Выставленные автоматизированной системой проверки баллы преподаватель, как финальный арбитр, может изменить. Баллы также могут варьироваться в зависимости от сроков сдачи (хранятся в метаинформации попытки), что также впоследствии сможет вручную скорректировать преподаватель.

Список попыток доступен студенту на странице задачи и преподавателю в статистике, что позволяет последнему изучать историю сдачи задания, узнавать проблемы, с которыми столкнулся студент, а студенту вернуться к одной из предыдущих версий решения [8].

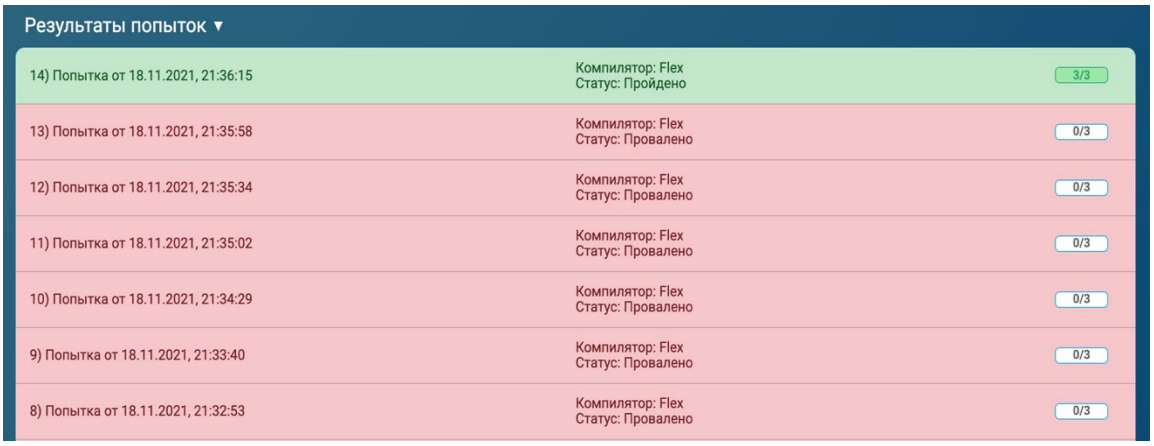

Рис. 2 Список попыток сдачи решения

3. Механизм черновиков

Среди списка попыток можно выделить еще один специальный уровень - текущий, названный здесь *черновиком*. Черновик характеризуется ассоциированной связью заданий со студентом, самим решением, визуализацией состояния решения на странице задачи для студента и у преподавателя в статистике. Черновик с точки зрения платформы также является попыткой, однако имеет специальный статус – *draft*. Эта попытка минует стадию верификации в тестирующей системе и сразу сохраняется в общий список попыток, где доступна для дальнейшего редактирования.

Такой подход предоставляет единый простой и естественный механизм доступа и хранения как для черновиков, так и для уже проверенных попыток, в общей парадигме хранения заданий студента, при этом черновики выделяются только специальным статусом.

При этом естественным является ограничение на число черновиков у студента для конкретного задания. Таким образом, учитываются особенности ЦОП Мирера, в которой существует возможность проверки задачи на нескольких языках программирования, различных вариантов сдачи задания студентом. Количество черновиков студента в одном задании полагается равным количеству доступных вариантов сдачи задания: так для задачи по программированию – это число доступных студенту компиляторов, а, например, в случае задачи на свободный ввод черновик является единственным.

При создании нового черновика в данном варианте сдачи старый должен удаляться, что для студента и преподавателя выглядит как существование единственного черновика, который имеет и содержимое, и обновленную метаинформацию.

Естественно, что при клиент-серверном решении в ЦОП Мирера черновик не может обновляться на сервере моментально и затратным будет решение в любой момент времени поддерживать полное соответствие между клиентом и сервером. При реализации черновика используются следующие приемы: каждое изменение в тексте решения студентом обрабатывается локально и при этом формируется запрос на создание нового черновика при помощи специальной функции, которая осуществляет посылку запроса на сервер о создании черновика. Данная функция обернута в специальный декоратор *debounce*, который откладывает ее выполнение, пока не истечет некий промежуток времени с момента предыдущей активности данной функции [9]. Вызов функции происходит с параметрами, которые были переданы в последней попытке ее вызова. В ЦОП Мирера в качестве данного промежутка используется интервал времени, равный половине секунды, что позволяет не нагружать систему обменами с сервером при постоянном наборе текста, а фактически выполнять сохранение только при прерывании процесса редактирования.

С точки зрения пользователя потеря изменений в случае сбоя или аварийного закрытия самим пользователем страницы с редактором без синхронизации с серверной частью будут ментально минимальными, вне зависимости от объема набранного текста, так как последние модификации текста находятся в точке внимания человека и легко восстановимы им самим при повторном заходе в систему в ближайшее время.

Декоратор *debounce* обладает специальным методом *flush*, который позволяет выполнить последнее отложенное выполнение функции, кото-

рый инициируется при покидании страницы редактора, что приводит к немедленному созданию черновика.

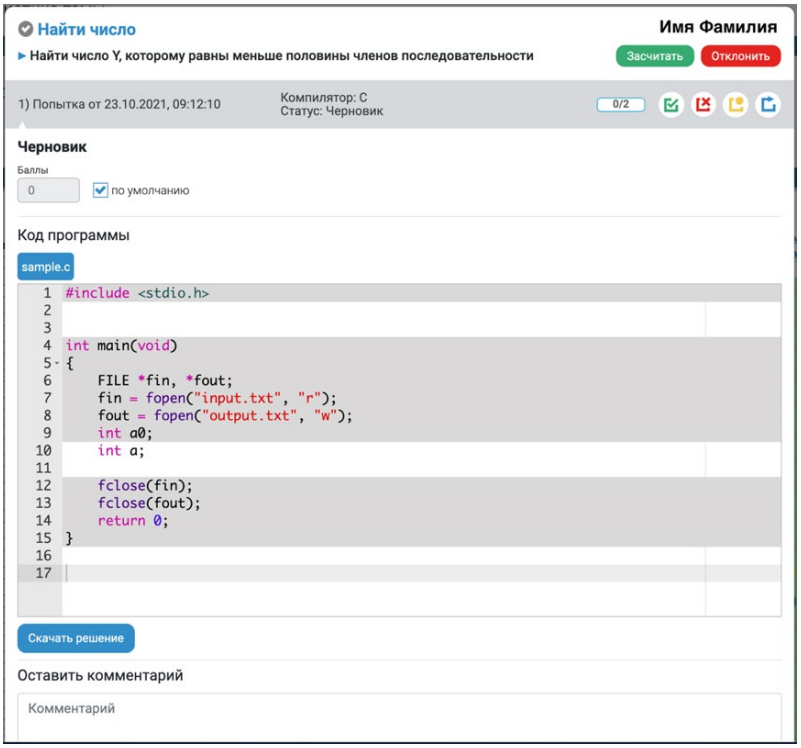

Рис. 3 Черновик студента в статистике группы

В ЦОП Мирера преподавателю доступна статистика студентов по всем задачам. В частности, педагогу виден список попыток ученика, в том числе черновики, которые являются попытками со специальным статусом. Поскольку черновики в любой момент времени содержат текущую версию решения, то преподаватель, фактически, может наблюдать за процессом написания решения студентом онлайн. Для этого не нужно обновлять страницу, достаточно открыть черновик студента и наблюдать за процессом редактирования учащимся своего решения.

4. Механизм веб-сокетов

С помощью аналогичного попыткам и черновикам механизма можно в режиме 24x7 динамически поддерживать коммуникацию между педагогом и слушателем курса. Так, с помощью активно применяющегося в ЦОП Мирера механизма веб-сокетов, можно осуществлять отправку сообщений и уведомлений пользователям, находящимся в режим онлайн, что позволяет исключить затратный механизм обновления страницы для получения актуальных данных. Отображение обновленных данных происходит автоматически при их изменении [10].

Каждый раз, когда студент отправляет решение или черновик, преподаватель получает сокет-уведомление об этом, что, в том числе, позволяет отображать актуальные данные в статистике группы. В свою очередь студент получает сокет-сообщение об обновленном статусе попытки и результатах автоматически протестированного решения, что позволяет увидеть ему обновленное состояние задания, находясь на странице сдачи задачи.

При этом преподаватель, как финальный арбитр, может изменять статус решения вручную, добавить комментарий к решению студента, засчитывать или отклонять решения. Информация об этих действиях поступает студенту через вебсокеты и также отображается онлайн, что предоставляет ученику возможность реагировать, исправляя ошибки решения, используя подсказки педагога [11].

ЦОП «Мирера» позволяет преподавателю осуществлять множественный вход в систему, что фактически снимает ограничения с процесса наблюдения за работой студентов в группе и лимитируется лишь размером и количеством экранов мониторов у преподавателя, а также его личными когнитивными способностями. ЦОП «Мирера» предоставляет педагогу возможность

в режиме реального времени наблюдать за работой всех студентов в группе одновременно, своевременно реагируя на ошибки и проблемы учащихся при решении заданий.

На схеме каждый из студентов получает информацию о тестировании его решения. При этом оба преподавателя получают данную информацию, поскольку связаны с группой, в которой состоят студенты. Также если «Преподаватель 1» оставит комментарий к решению «Студента 2», то «Студент 2» моментально получит информацию об этом благодаря механизму вебсокетов, внедренному в ЦОП Мирера [12].

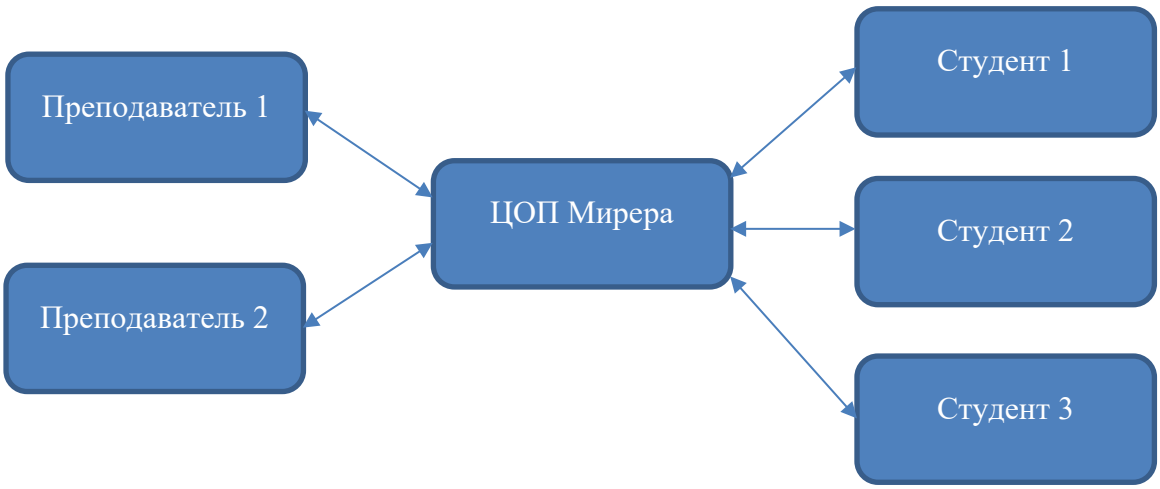

Рис. 4 Схема взаимодействия пользователей

5. Заключение

Черновики, как отдельный механизм, позволяют пользователям (студентам и преподавателям) в случае различных сбоев в связи с сервисами ЦОП Мирера не потерять набранный текст в задании, сохраняя его практически при каждом изменении текста решения, даже если решение не было отправлено на проверку. Таким образом, пользователь может начать выполнение задания, прервать свою работу и вернуться к нему позднее, поскольку черновик будет хранить информацию о текущем прогрессе и позволит продолжить работу с того места, где пользователь остановился в предыдущий раз. При этом механизм черновиков позволяет предотвратить случайный уход со страницы путем обновления или закрытия страницы с ЦОП Мирера.

Используя черновики, преподаватель может контролировать процесс написания решения студентами заданий в режиме реального времени, что также позволяет ему активнее участвовать в образовательном процессе, фактически, видя работу всей группы одновременно.

Механизм черновиков может стать центральным элементом прокторинга при сдаче контрольных, промежуточных и итоговых испытаний студентами. В процессе работы студента в рамках его цифрового следа формируется почерк решения студентом заданий. К этому относятся временные характеристики стиля написания, для текстовых ответов – набор характерных слов, фраз, для редактирования – стиль изменения текста, используемые управляющие сочетания клавиш, поведение в стрессовых ситуациях (на контрольных), характерные ошибки и многое другое. Анализ собранной информации описывает поведение студента, формируя модель цифрового двойника слушателя курсов, которую можно сравнить с поведенческой моделью того же студента на испытаниях, а детектирование наличия существенных различий между моделями поведения будет говорить педагогу о возможном факте выполнения итоговых испытаний другим человеком.

Исследование выполнено в рамках государственного задания по проведению фундаментальных исследований по теме «Разработка, реализация и внедрение семейства интегрированных многоязыковых сред программирования с автоматизированной проверкой заданий для учащихся образовательных организаций, ДОО, младшей, основной и старшей школы и студентов педагогических университетов.» (№ 0580-2021-0010).

The Mechanism of Interaction of a Teacher with Students in the Digital Educational Platform of Mirera

Alexander Leonov, Kirill Mashchenko, Anton Orlovskii, Artem Shlyakhov

Abstract. The creation of digital educational platforms and environments gives educators the opportunity to intensify the learning process by simplifying and expanding the ways of extracurricular communication with students. On the other hand, the student is given the opportunity to master the competencies being studied, send assignments for automatic verification, in a convenient form at any time, if there is a connection with a digital educational platform. Platform developers, in turn, must provide a convenient and reliable way to support this interaction. The article describes a solution to this problem using the example of the author's digital educational platform Mirera. The possibility of using such a mechanism to implement an alternative approach to proctoring is noted.

Keywords: digital educational platform, digital educational environment, Mirera, web-sockets, drafts

Литература

1. Бесшапошников Н.О., Дьяченко М.С., Леонов А.Г., Мащенко К.А. Использование элементов искусственного интеллекта в виртуальных помощниках преподавателя // Успехи кибернетики, 2020, том 1, №3, с. 15-22.

2. Леонов А.Г., Орловский А.Е. Методы интеграции цифровых образовательных сред в цифровую образовательную платформу Мирера // Труды НИИСИ РАН. – 2021. – Т. 11, №3. – С. 59-65.

3. Платформа Мирера [Электронный ресурс] URL: https://www.mirera.ru (дата обращения: 01.12.2021).

4. Ace Editor [Электронный ресурс] URL: https://ace.c9.io/ (дата обращения: 01.12.2021).

5. Прокин А.А., Рузманов А.А. Использование веб-сокетов в Интернет-приложениях // E-Scio 2019.

6. I. Fette, A. Melnikov WebSocket URIs // RFC 6455 The WebSocket Protocol. IETF. Sec.3.

7. Бесшапошников Н.О., Дьяченко М.С., Леонов А.Г., Матюшин М.А., Орловский А.Е. Использование машинного обучения и нейронных сетей для автоматической верификации заданий в текстовом и графическом представлении и помощи преподавателю // Успехи Кибернетики. 2020, том 1, №2, с. 39-45.

8. Бетелин В.В., Кушниренко А.Г., Семенов А.Л., Сопрунов С.Ф. О цифровой грамотности и средах ее формирования // Информатика и ее применения, том 14, %4, с. 102-109.

9. Lodash utility library [Электронный ресурс] URL: https://www.lodash.com (дата обращения: 01.12.2021).

10. Царева Е.В. Разрешение конфликтных ситуаций при синхронизации многопользовательских online-приложений // Вестник Томского государственного университета. Управление, вычислительная техника и информатика.

11. Леонов А.Г., Первин Ю.А. Качественные оценки эффективности методики обучения элементам информатики в пропедевтическом курсе // Ярославский педагогический вестник, №5, с. 92-96.

12. Шестаков В.С., Сагидуллин А.С. Применение технологии WebSocket в Web-приложениях технологического назначения // Приборостроение, 2015.

Подписано в печать 22.12.2021 г.
Формат 60x90/8
Печать цифровая. Печатных листов 12,5
Тираж 100 экз. Заказ № 1066

Отпечатано в ППП «Типография «Наука»
121099, Москва, Шубинский пер., 6