ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛЕНИЯ В НАУКЕ, ТЕХНИКЕ И ОБРАЗОВАНИИ

УДК 004.4’4, 004.272

МЕТОДИКА АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ ЗАДАЧ НА ТЕМУ АНАЛИЗА НИЗКОУРОВНЕВЫХ ПРЕДСТАВЛЕНИЙ ПРОГРАММНОГО КОДА

А. В. Горчаков, И. И. Фандеев

МИРЭА – Российский технологический университет

Аннотация. В статье представлена методика автоматической генерации учебных задач на тему анализа байткода виртуальных машин, а также программное средство на основе методики. Актуальность работы обусловлена потребностью в формировании у студентов направления подготовки 09.04.02 «Программная инженерия» компетенций, связанных с отладкой, анализом и обратной разработкой программных систем. Предложенный в статье подход основан на методике автоматической генерации задач по программированию П. Н. Советова, для порождения задач применяется метод «сгенерировать и проверить» из области программирования в ограничениях и техники трансляции и интерпретации сгенерированных промежуточных представлений. Задачи генерируются в виде тестов с поддержкой автоматической проверки решений в виде короткого ответа. Приведены оценки производительности разработанных генераторов задач, а также результаты их внедрения в учебный процесс.

Ключевые слова: автоматическая генерация задач, низкоуровневые представления, виртуальные машины, дизассемблирование, трансляторы, программирование в ограничениях.

Введение

В современных условиях способность разработчика программного обеспечения (ПО) анализировать низкоуровневое представление программного кода, такое как, например, байткод языковых виртуальных машин (ВМ) Lua [1] и CPython [2] или байткод кроссплатформенных ВМ JVM (Java Virtual Machine) [3] и CLR (Common Language Runtime) [4], становится одной из ключевых компетенций, необходимых для решения широкого круга практических задач: от отладки ПО с целью внесения в него исправлений для обеспечения корректной работы программной системы до анализа защищённости ПО или обратной разработки унаследованных программных систем.

Однако, формирование этой компетенции у студентов при обучении программированию и, в частности, при обучении отладке и обратной разработке ПО, в настоящее время затруднено ввиду недостатка вариативных учебных заданий — решения задач, создаваемые преподавателями вручную, быстро оказываются в открытом доступе и покрывают лишь частные случаи низкоуровневых представлений кода. Вопросы автоматической генерации заданий для использования в учебном процессе дисциплин высшей школы рассматривались в ряде работ отечественных и зарубежных авторов. В частности, в [5] представлен генератор тестовых заданий по дифференциальным уравнениям, в [6] рассмотрен генератор тестовых заданий на тему разрешения зависимостей программных пакетов и генератор задач на работу с системой контроля версией, в [7] описывается подход к генерации заданий на тему определения результатов выполнения программ, сгенерированных по правилам из порождающей контекстно-свободной грамматики (КСГ).

Предлагаемая в рамках данной работы методика автоматической генерации задач направлена на решение проблемы недостатка вариативных учебных заданий на тему анализа низкоуровневых представлений программного кода. Методика является модифицированной версией методики автоматической генерации учебных задач по программированию с поддержкой автоматической проверки их программных решений, предложенной П. Н. Советовым в работе

[8] и основанной на применении метода решения задач программирования в ограничениях «сгенерировать и проверить», техник трансляции и интерпретации сгенерированных промежуточных представлений (ПП) задач.

1. Методика и программное средство для её практического применения

Предлагаемая методика автоматической генерации задач на тему анализа низкоуровневых представлений программного кода включает следующие шаги:

1. Определение структуры задач в виде, например, порождающей КСГ.
2. Формализация ограничений для порождаемых задач в виде булевой формулы.
3. Автоматическая генерация ПП задач, удовлетворяющих ограничениям, на основе метода решения задач программирования в ограничениях «сгенерировать и проверить».
4. Трансляция сгенерированных ПП в программный код на высокоуровневом языке программирования и компиляция кода в целевое низкоуровневое представление.
5. Интерпретация сгенерированных ПП для вычисления результата выполнения целевого низкоуровневого представления программного кода.
6. Загрузка низкоуровневых представлений программного кода и результатов их выполнения в систему управления обучением.

Предложенная методика была применена для автоматизации некоторых видов рутинной деятельности преподавателей по подготовке сборников задач для дисциплины «Разработка кроссплатформенных программных систем», которая преподаётся студентам направления подготовки 09.04.02 «Программная инженерия» Института информационных технологий РТУ МИРЭА. Архитектура разработанного на основе предложенной методики программного средства, выполняющего порождение тестовых заданий на тему анализа текстового формата байткода ВМ Lua [1], CPython [2], JVM [3] и CLR [4] представлена на рис. 1.

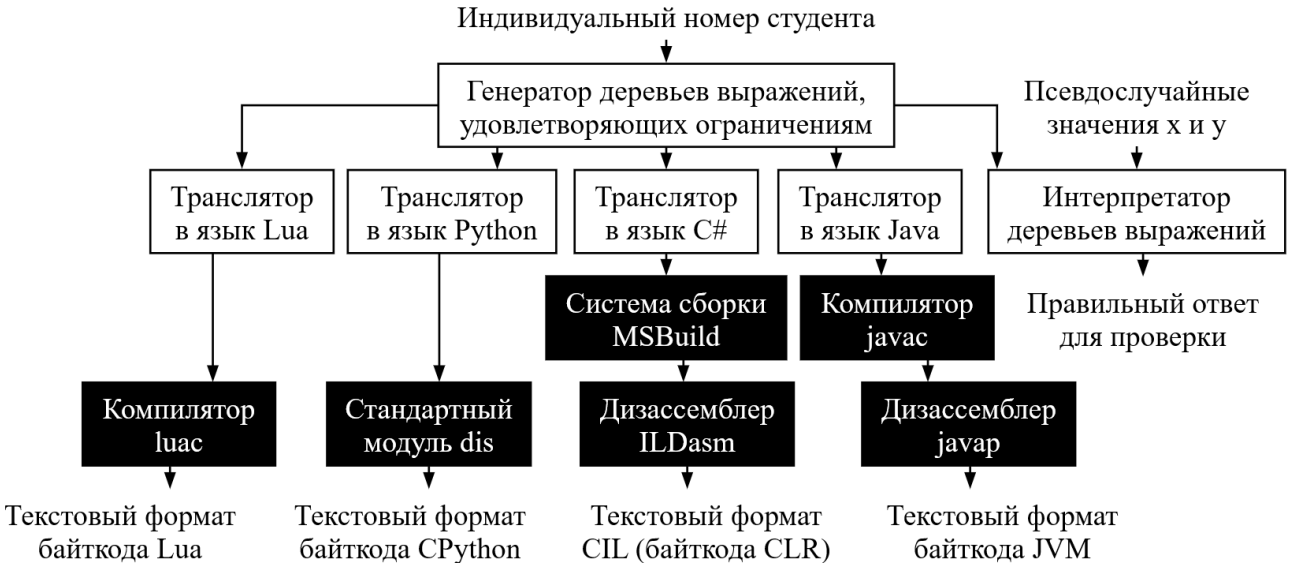

Рис. 1. Архитектура генератора задач на тему анализа низкоуровневых представлений кода

Чёрные прямоугольники на рис. 1 соответствуют сторонним модулям, необходимым для преобразования ПП в текстовый формат байткода ВМ. Белые прямоугольники на рис. 1 соответствуют внутренним компонентам разработанного программного средства. Как показано на рис. 1, в разработанном программном средстве ПП — это деревья выражений, их генерация выполняется на основе порождающей КСГ по следующим правилам:

$$C \to 1|2|3|4|5, \tag{1}$$

$$\mathrm{V} \to \mathrm{x} \mid \mathrm{y}, \tag{2}$$

$$\mathrm{O} \to -\mid * \mid / \mid \% \mid \gg \mid \ll \mid \&, \tag{3}$$

$$\mathrm{E} \to (\mathrm{V}\ \mathrm{O}\ \mathrm{C}) \mid \mathrm{V} \mid \mathrm{C}, \tag{4}$$

$$\mathrm{P} \to + \mid -, \tag{5}$$

$$\mathrm{S} \to (((\mathrm{E}\ \mathrm{P}\ \mathrm{E})\mathrm{P}\ \mathrm{E})\mathrm{P}\ \mathrm{E}). \tag{6}$$

Генерация деревьев выражений по КСГ начинается с правила (6). Символ | используется для компактной записи альтернатив, а нетерминальные символы обозначены большими буквами. Порождаемые выражения (4) могут содержать имена переменных x и y (2), целочисленные константы (1), арифметические или побитовые операции (3). Несколько выражений (4) объединяются в одно (6) при помощи операторов сложения или вычитания (5).

После генерации ПП в виде дерева выражения G по КСГ, начиная с правила (6), выполняется проверка удовлетворения ПП G следующего ограничения:

$$\left(\exists n \in V(G) : n \in \{\mathrm{x}, \mathrm{y}\}\right) \wedge \left(\exists n \in V(G) : n \in \{\gg, \ll, \&, \%\}\right), \tag{7}$$

где $V(G)$ — множество вершин ПП задачи G, n — вершина дерева G, x и y — имена переменных, согласно (7) хотя бы один из символов x и y должен присутствовать в G.

В случае несоответствия ПП ограничению (7) процесс генерации ПП по КСГ (6) и проверки ограничения (7) повторяется до достижения успеха.

2. Трансляторы промежуточных представлений в текстовый формат байткода

На следующем этапе выполняется трансляция ПП в программный код на высокоуровневом языке программирования, который затем компилируется в низкоуровневое представление. Детали реализации трансляторов ПП в текстовый формат байткода языковых ВМ Lua [1] и CPython [2] приведены в табл. 1.

Таблица 1

Детали реализации трансляторов ПП в байткод Lua и CPython

ВМ	Детали реализации	Пример байткода в текстовой форме

Lua	Компиляция и дизассемблирование:
	`$ luac -l -l program.lua`
	
	Содержимое файла program.lua:
	`function main(x, y)`
	`  return ((1 + x) - y) + 2`
	`end`
	
	Необходимые компоненты:
	`Lua 5.3.6.`
		`1 [2] GETTABUP 0 0 -1`
		`2 [2] ADD 0 -2 0`
		`3 [2] GETTABUP 1 0 -3`
		`4 [2] SUB 0 0 1`
		`5 [2] ADD 0 0 -4`
		`6 [2] RETURN 0 2`
		`constants (4) for 00:`
		`1 «x»`
		`2 1`
		`3 «y»`
		`4 2`

CPython	Компиляция и дизассемблирование:
	`$ python program.py`
	
	Содержимое файла program.py:
	`import dis`
	`dis.dis('((1 + x) - y) + 2')`
	
	Необходимые компоненты: Python `3.12.0.`
		`2 LOAD_CONST 0 (1)`
		`4 LOAD_NAME 0 (x)`
		`6 BINARY_OP 0 (+)`
		`10 LOAD_NAME 1 (y)`
		`12 BINARY_OP 10 (-)`
		`16 LOAD_CONST 1 (2)`
		`18 BINARY_OP 0 (+)`
		`22 RETURN_VALUE`

Для генерации постановок задач на тему анализа байткода ВМ, являющихся частью интерпретаторов языков Lua и Python, сгенерированное ПП, удовлетворяющее ограничению (7), транслируется в программный код на языке Lua или Python, после чего код на языке Lua дизассемблируется при помощи компилятора luac [1] с указанными в табл. 1 опциями, а код на языке Python дизассемблируется при помощи модуля dis, встроенного в интерпретатор CPython [2], посредством исполнения кода на языке Python, показанного в табл. 1.

Детали реализации трансляторов ПП в текстовый формат байткода кроссплатформенных ВМ JVM [3] и CLR [4] приведены в табл. 2.

Таблица 2

Детали реализации трансляторов ПП в байткод JVM и CLR

ВМ	**Детали реализации**	**Пример байткода в текстовой форме**

CLR	Компиляция и дизассемблирование:
	`$ dotnet build -c Debug`
	`$ ildasm /caverbal /item:»Src.Program::Main» bin/Debug/net8.0/Src.dll /out:Program.il`
	
	Содержимое файла Program.cs:
	`namespace Src;`
	`public class Program {`
	`  public int Main(`
	`  int x, int y) {`
	`  return ((1 + x) - y) + 2;`
	`  }`
	`}`
	
	Необходимые компоненты:
	.NET 8.0.406,
	ILDAsm 9.0.0.
		`.class public auto ansi beforefieldinit Src.Program extends System.Object {`
		`  .method public hidebysig instance int32 Main(int32 x, int32 y) cil managed {`
		`  .maxstack 2`
		`  .locals init (int32 V_0)`
		`  IL_0000: nop`
		`  IL_0001: ldc.i4.1`
		`  IL_0002: ldarg.1`
		`  IL_0003: add`
		`  IL_0004: ldarg.2`
		`  IL_0005: sub`
		`  IL_0006: ldc.i4.2`
		`  IL_0007: add`
		`  IL_0008: stloc.0`
		`  IL_0009: br.s IL_000b`
		`  IL_000b: ldloc.0`
		`  IL_000c: ret`
		`  }`
		`}`

JVM	Компиляция и дизассемблирование:
	`$ javac Program.java`
	`$ javap -c Program.class`
	
	Содержимое файла Program.java:
	`public class Program {`
	`  public static int main(`
	`  int x, int y) {`
	`  return ((1 + x) - y) + 2;`
	`  }`
	`}`
	
	Необходимые компоненты: Java 21.0.8.
		public class Program {
		public static int main(int,int);
		Code:
		0: iconst_1
		1: iload_0
		2: iadd
		3: iload_1
		4: isub
		5: iconst_2
		6: iadd
		7: ireturn
		}

Для генерации задач на тему анализа байткода JVM ПП сначала транслируется в код на языке Java, сохраняемый в файл Program.java. После этого при помощи компилятора javac генерируется байткод JVM и сохраняется в файл Program.class, который затем дизассемблируется утилитой

javap. Для генерации задач на тему анализа байткода CLR ПП транслируется в код на языке C#, сохраняемый в файл Program.cs, после чего выполняется сборка .NET-проекта при помощи системы сборки MSBuild. После сборки файл с именем Src.dll, содержащий байткод на языке CIL (Common Intermediate Language), дизассемблируются при помощи утилиты ILDasm [9].

Результаты генерации задач на тему анализа текстового формата байткода языковых и кроссплатформенных ВМ были преобразованы в формат GIFT и затем загружены в систему управления обучения Moodle [6]. Пользовательский интерфейс Moodle, который видит студент, выполняющий задачу на тему анализа CIL – байткода CLR, показан на рис. 2.

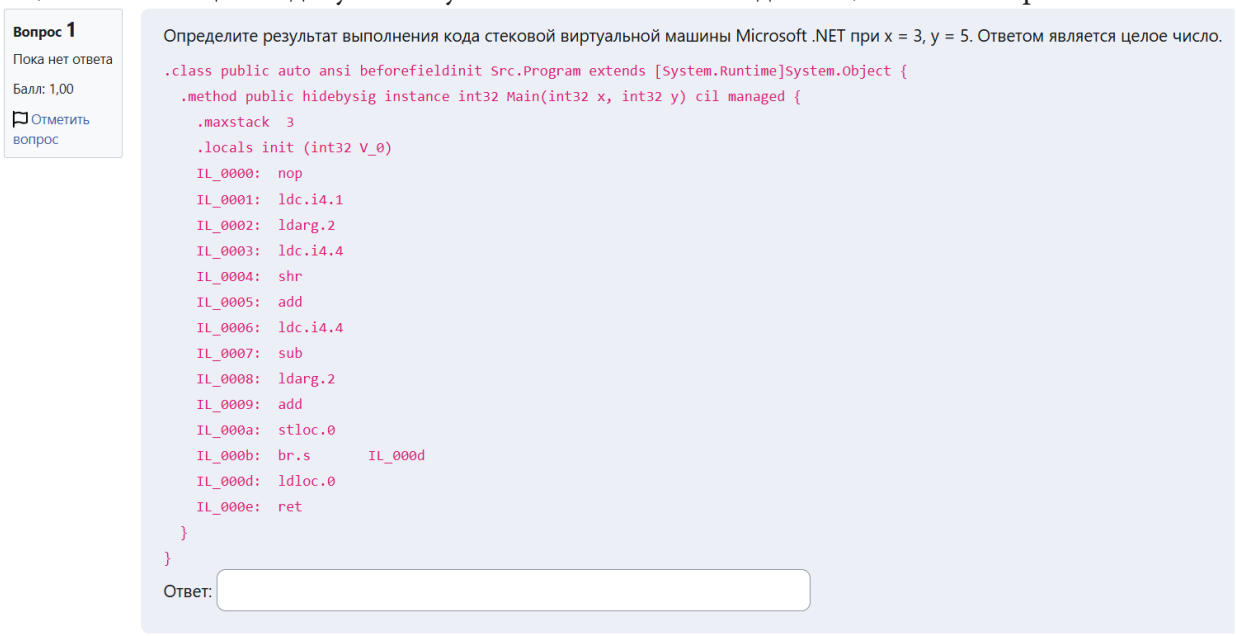

Рис. 2. Задача на тему анализа байткода CLR в пользовательском интерфейсе Moodle

Как показано на рис. 2, подстановка задачи в системе управления обучением содержит текстовый формат байткода ВМ, входные значения переменных, а также ссылку на документацию по ВМ, результат выполнения байткода которой необходимо определить студенту. Интерпретатор ПП задачи, который является частью разработанного программного средства для генерации задач (см. рис. 1), вычисляет значение сгенерированного ПП и сохраняет полученное значение в систему управления обучением для поддержки автоматической проверки ответа, указанного студентом.

3. Оценки производительности генераторов

Оценка производительности разработанного программного средства, применяемого для генерации учебных задач на тему анализа низкоуровневых представлений программного кода для 4-х различных ВМ проводилась на компьютере со следующими характеристиками: Intel Core i3-1115G4, 8 GB RAM, Windows 11. Программное средство было реализовано на языке программирования Python версии 3.12.0. Взаимодействие с утилитами командной строки, такими как компиляторы luac, javac, система сборки MSBuild, дизассемблеры javap и ILDasm, осуществлялось при помощи стандартного модуля subprocess языка Python.

Полученные результаты приведены в табл. 3.

Таблица 3

Оценки производительности генераторов задач различных типов

ВМ	Lua	CPython	CLR	JVM
Производительность, задач в минуту	1787	977	37	74

Как показано в таблице 3, генераторы задач на тему анализа байткода ВМ языков Lua и CPython работают быстрее, чем генераторы задач на тему анализа байткода CLR и JVM. Это связано с тем, что байткод ВМ Lua и CPython существенно проще, а компиляция и дизассемблирование программ на этих языках в разработанном программном средстве не требует сохранения байткода на диск [1, 2]. Кроме того, байткод JVM и CLR предназначен не только для интерпретации, но и для оптимизации на этапе выполнения Just-In-Time (JIT)-компилятором, что требует сохранения детальной информации о типах, сигнатурах методов, наследовании и других высокоуровневых свойствах программы [3, 9].

4. Результаты внедрения в учебный процесс

Разработанное на основе предложенной методики программное средство использовалось для генерации и загрузки в систему управления обучением по 500 случайных задач на тему анализа байткода ВМ Lua, CPython, JVM, CLR. На основе полученного пула из 2000 задач было создано 2 теста. Первый тест включал 12 случайно выбранных из пула задач на тему определения результата выполнения байткода языковых ВМ Lua и CPython. Второй тест включал 12 случайно выбранных из пула задач на тему определения результата выполнения байткода кроссплатформенных ВМ JVM и CLR.

В случае сдачи этих тестов не на максимальный балл студенты могли попытаться решить задачи снова, до исчерпания лимита в 7 попыток, при этом использовался метод оценивания «высшая оценка». Для успешной сдачи теста студентам было предложено реализовать упрощённые эмуляторы стековой и регистровой ВМ. Сведения о результатах выполнения этих тестов студентами дисциплины «Разработка кроссплатформенных программных систем», которая преподаётся студентам Института информационных технологий РТУ МИРЭА, приведены в табл. 4.

Таблица 4

Сведения о выполнении сгенерированных задач студентами

Метрика	ВМ Lua и CPython	ВМ CLR и JVM
Всего попыток	493	187
Задач в тесте	12	12
Среднее число решённых задач	9	10
Медианное число решённых задач	10	11
Среднее время выполнения, мин.	27	22
Медианное время выполнения, мин.	25	18
Среднее число попыток	2	2
Медианное число попыток	2	1

Как показано в табл. 4, среднее и медианное число решённых задач во 2-м тесте выше, а медианное число попыток — ниже, как и время выполнения теста. Это связано, с одной стороны, с тем, что 2-й тест предлагался к выполнению в течение семестра после 1-го теста и к моменту начала работы над 2-м тестом студенты уже были знакомы с задачами на тему анализа байткода ВМ. Разница в результатах связана ещё и с тем, что 1-й тест содержал задачи на тему анализа байткода стековой ВМ CPython и регистровой ВМ Lua, а ВМ CLR и JVM из 2-го теста обе являются стековыми.

Заключение

В работе представлена методика и программное средство для автоматической генерации учебных задач на тему анализа низкоуровневых представлений программного кода, таких как байткод BM Lua, CPython, CLR и JVM. Предложенная методика основана на результатах, полученных в работе [8], и предполагает применение метода решения задач программирования в ограничениях «сгенерировать и проверить», порождающей КСГ, трансляторов и интерпретатора ПП задач для поддержки автоматической проверки решений. Полученные оценки производительности и результаты внедрения в учебный процесс подтверждают применимость предложенной методики для автоматизации некоторых видов рутинной деятельности преподавателей высшей школы по подготовке сборников задач.

Дальнейшая работа может быть направлена на добавление поддержки трансляции ПП задач и в другие низкоуровневые представления программного кода, такие как, например, деревья абстрактного синтаксиса, LLVM IR (Low Level Virtual Machine Intermediate Representation), WAT (WebAssembly Text Format), формат команд процессорных архитектур x86_64, ARM, RISC-V.

Литература

1. *Ierusalimschy R.* The Implementation of Lua 5.0 / R. Ierusalimschy, L.H. De Figueiredo, W. Celes Filho // Journal of Universal Computer Science. – 2005. – Vol. 11, № 7. – P. 1159–1176.

2. *Chen Z.* Hybrid information flow analysis for python bytecode / Z. Chen, L. Chen, B. Xu // Proceedings of the 2014 11th Web Information System and Application Conference. – IEEE, 2014. – P. 95–100.

3. *Lindholm T.* The Java Virtual Machine Specification. Java SE 7 Edition / T. Lindholm, F. Yellin, G. Bracha, A. Buckley // Addison-Wesley, 2013. – P. 606.

4. *Макаров А. В.* Common Intermediate Language и системное программирование в Microsoft .NET / А. В. Макаров, С. Ю. Скоробогатов, А. М. Чеповский // М.: Национальный Открытый Университет ИНТУИТ, 2016. – 398 с.

5. *Муханова А. А.* Разработка генератора тестовых заданий по дифференциальным уравнениям для системы дистанционного обучения Moodle / А. А. Муханова, С. А. Муханов, А. И. Нижников // Вестник Российского университета дружбы народов. Серия: Информатизация образования. – 2014. – № 3. – С. 100–107.

6. *Советов П. Н.* Модели и алгоритмы автоматической генерации учебных задач по работе с системой git и разрешению зависимостей между пакетами / П. Н. Советов, А. В. Горчаков // Электронный научный журнал ИТ-Стандарт. – 2025. – № 3. – С. 47–57.

7. *Ade-Ibijola A.* Syntactic Generation of Practice Novice Programs in Python / A. Ade-Ibijola // Annual Conference of the Southern African Computer Lecturers' Association. – Springer, 2018. – P. 158–172.

8. *Sovietov P. N.* Automatic Generation of Programming Exercises / P. N. Sovietov // Proceedings of the 2021 1st International Conference on Technology Enhanced Learning in Higher Education (TELE). – IEEE, 2021. – P. 111–114.

9. *Lidin S.* Expert .NET 2.0 IL Assembler / S. Lidin // Apress, 2007. – 530 с.

УДК 621–32

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ТЕХНОЛОГИЙ АВТОМАТИЗАЦИИ СЛОЖНЫХ ПРОИЗВОДСТВЕННЫХ КОМПЛЕКСОВ

Лавлинская О. Ю., Сангаре Умар

Воронежский государственный университет

Аннотация. В статье проводится сравнительный анализ современных технологий автоматизации сложных производственных комплексов, таких как нефтеперерабатывающие заводы, металлургические комбинаты, предприятия химической промышленности и фармацевтики. Рассматриваются ключевые архитектурные подходы — от традиционных систем диспетчерского управления и сбора данных (SCADA) и распределенных систем управления (DCS) до современных концепций Industrie 4.0, киберфизических систем (CPS) и платформ Industrial Internet of Things (IIoT). Критериями для сравнения выступают масштабируемость, гибкость, интеграционная способность, безопасность, стоимость владения и поддержка сквозной цифровизации (Digital Thread). По результатам анализа сформулированы рекомендации по выбору технологической платформы в зависимости от специфики и стратегических целей предприятия.
Ключевые слова: автоматизация, SCADA, DCS, IIoT, Индустрия 4.0, киберфизические системы, цифровой двойник, гибкое производство, сравнительный анализ.

Введение

Современные промышленные производства характеризуются сложностью, высоким уровнем взаимосвязанности технологических процессов и жесткими требованиями к эффективности, качеству и безопасности производственных процессов. Устаревшие системы автоматизации, функционирующие по принципу «жесткой» логики, не способны удовлетворить растущие потребности в адаптивности, оптимизации ресурсов и скорости вывода новых продуктов на рынок. На смену им приходит новое поколение технологий, базирующихся на симбиозе виртуальных моделей и физических процессов. Цель данного анализа — систематизировать существующие решения в области промышленной автоматизации, выявить их сильные и слабые стороны, а также определить эффективные подходы к выбору технологий автоматизации и цифровизации для решения производственных задач.

1. Обзор мирового рынка промышленной автоматизации и цифровизации

По данным аналитического обзора [1] размер мирового рынка промышленной автоматизации в 2021 году оценивался в 177,57 млрд долларов, прогноз на 2030 год составляет 441,7 млрд долларов, что показано на рис. 1. В 2021 году оставная доля в 34 % высокотехнологичных про-

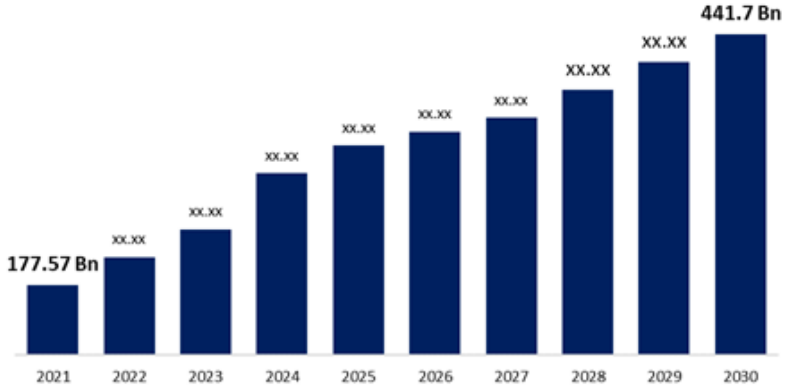

Рис.1. Динамика роста мирового рынка производственной автоматизации

изводств приходилась на Северную Америку, но лидерами рынка к 2030 будут страны Азиатско-Тихоокеанский региона, так как тенденции роста в этих странах выше общемировых показателей. Среднегодовой прирост (CAGR) суммы вложений с 2022 по 2030 год в промышленную автоматизацию и цифровизацию составит 8,84 %.

В РФ на период 2025 года рынок автоматизации и цифровизации составляет 250 млн. долл, прогнозируется среднегодовой рост высокотехнологичных решений на 5,1 %. Темп роста ниже мирового уровня, но, достаточно высокий для решения критически важных задач в сфере промышленной автоматизации [1]. Ключевые драйверы рынка промышленной автоматизации в РФ на 2025-2026 год:

1. Умные производственные линии и роботизация. В данном сегменте доля компаний, внедривших роботов в производство, выросла до 45 %. Данный показатель выше предыдущих периодов, но существенно отстает от мировых тенденций. Лидером роста робототизированных производств является Китай. В 2025 году в Китае был установлен мировой рекорд — 2 027 000 промышленных роботов, работающих на заводах. В 2024 г. ежегодные установки достигли 295 000 единиц. Этот показатель вырос 7 %, что является самым высоким показателем за всю историю наблюдений и составляет 54 % мирового спроса в области робототехники. Об этом говорится в отчете World Robotics 2025, представленном Международной федерацией робототехники (IFR) [2].

2. Индустриальный интернет вещей (IIoT). В 2025 году число подключённых IIoT-устройств на российских предприятиях достигло уровня в 50 млн. экземпляров. Оценка устройств IIoT в мировом масштабе на 2025 год – 75 миллиардов подключенных устройств [1].

3. Big Data и предиктивная аналитика. Внедрение интеллектуальных технологий позволяют сокращать затраты на техническое обслуживание, управление и контроль качества на 30 %.

Сегмент внедрения технологий больших данных в сочетании с алгоритмами прогнозирования – наиболее интенсивно развивающая отрасль автоматизации.

В целом, по данным агентства Infinium Global Research [3] российский рынок аналитики данных вырастет с 3 204,23 млн долларов США в 2024 году до примерно 13 963,17 млн долларов США к 2032 году, демонстрируя впечатляющий среднегодовой темп роста в 20,96 % в период с 2025 по 2032 год. Этот рост отражает растущую интеграцию процессов принятия решений на основе данных в различных отраслях, в том числе, в производстве.

Рассмотрим основные технологии автоматизации в промышленности.

2. Распределенные системы управления (DCS)

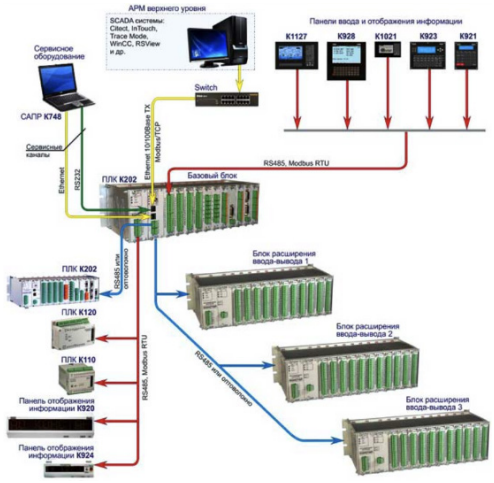

Рис. 2. Типовая архитектура распределенной системы управления

Централизованная многоуровневая архитектура для управления технологическими процессами в реальном времени объединяет контроллеры, системы ввода-вывода и HMI в единую экосистему с высокой надежностью. Примеры: Siemens SIMATIC PCS 7, Emerson DeltaV, ABB Ability™ System 800xA, а также российские решения от «Чинт Электрик», НИИ «Энергосетьпроект», РУСЭНЕРГОСБЫТ и др. [4–8]. Типовая архитектура распределенной системы управления представлена на рис. 2.

Основные преимущества распределенных систем управления: высокая надежность и предсказуемость, глубокая интеграция обо-

рудования, идеальная поддержка сложных ПИД-регуляторов, сильная встроенная функциональная безопасность [9].

Основные недостатки: «Закрытость» экосистемы, высокая стоимость лицензий и обновлений, сложность интеграции с оборудованием сторонних производителей, сложность модернизации.

Текущий объем рынка распределенных систем управления составляет $21,04 млрд (оценка на 2025 год). Ожидается, что к 2035 году рынок увеличится до $39,49 млрд, демонстрируя среднегодовой темп роста около 6,5 %.[2]

Важнейшими драйверами роста являются быстрое распространение технологий Интернет-вещей (IoT), автоматизация промышленного производства и необходимость эффективного управления электроэнергией.[3]

3. Системы диспетчерского управления и сбора данных (SCADA)

Системы, ориентированные на сбор данных с удаленных объектов (полевых контроллеров, PLC) и предоставление оператору средств для диспетчерского управления. Более гибкие, чем DCS, так как могут агрегировать данные из разнородных источников [10].

Примеры SCADA-систем:

Мировой рынок: Ignition, WinCC, GE Digital iFIX, Wonderware System Platform.

Российский рынок: КАСКАД (универсальная, высокая производительность), Альфа Платформа (гибкая, масштабируемая), МастерСКАДА 4D (модульная, визуализация), REDKIT SCADA 2.0 (гибкость, нефтегаз/металлургия), ИнтраСКАДА (интеграция с ERP, поддержка отечественного оборудования).

Основные преимущества систем диспетчерского управления и сбора данных — это гибкость и открытость (поддержка множества протоколов связи — OPC UA, MQTT), относительно низкая стоимость внедрения, отличная приспособленность для мониторинга распределенных активов (трубопроводы, электросети).

Основные недостатки систем диспетчерского управления и сбора данных: слабая, по сравнению с DCS, «глубина» управления процессами в реальном времени; сложности интегрирации с системами безопасности.

4. Платформы Промышленного Интернета Вещей (IIoT), концепция Индустрии 4.0

IIoT-платформы собирают данные с оборудования с помощью облачных технологий, обеспечивают аналитику и переход к предиктивному управлению. К достоинствам относятся масштабируемость, аналитика и поддержка цифровых двойников [11].

Недостатки: зависимость от сети, энергозатраты, киберриски и высокая стоимость обработки данных. Сравнение технологических подходов к автоматизации представлено в табл. 1.

Цифровые двойники объединяют автоматизацию производства с телекоммуникационными и инженерными технологиями, позволяя создавать виртуальные модели промышленных объектов. Производственный цикл реализуется в виртуальном пространстве по принципам MBSE, а затем интегрируется в физическую среду. Такой подход требует модульной архитектуры и автоматизации промышленных производственных комплексов [12,13].

5. Практические рекомендации по выбору технологий автоматизации промышленных комплексов

Выбор технологий зависит от задач предприятия: для крупных непрерывных производств подходит DCS с дополнением IIoT-платформой; для дискретных и гибридных — SCADA-систе-

Таблица 1

Сравнение технологий автоматизации промышленных производств

Критерий	DCS	SCADA	IIoT / Индустрия 4.0
Основная задача	Надежное управление непрерывным процессом в реальном времени	Сбор данных и диспетчеризация распределенных объектов	Сквозная аналитика, предиктивная аналитика, гибкость
Архитектура	Закрытая, централизованная, иерархическая	Более открытая, клиент-серверная	Децентрализованная, облачная, сервис-ориентированная
Интеграция	Глубокая, но в рамках экосистемы вендора	Гибкая, со множеством сторонних устройств (через OPC и т.д.)	Максимально открытая, горизонтальная интеграция IT/OT
Масштабируемость	Сложная и дорогая, в рамках одной экосистемы	Относительно простая, но может требовать добавления серверов	Практически неограниченная, за счет облачных технологий
Гибкость и адаптивность	Низкая, изменения требуют сложных процедур	Средняя, конфигурация изменяема	Высокая, возможность быстрого развертывания новых сервисов
Аналитика и AI/ML	Базовая, исторический тренд	Средняя, расширенная визуализация и отчетность	Высшая, предиктивное моделирование, оптимизация с помощью ML
Кибербезопасность	Сильная, но в рамках «закрытого» периметра	Зависит от реализации, требует постоянного контроля	Критически важна, требует комплексного подхода (zero trust и др.)
Стоимость владения (TCO)	Высокая начальная стоимость, высокие затраты на обновление	Умеренная начальная стоимость	Модель подписки (SaaS), но могут быть высоки затраты на интеграцию и данные

мы с PLC плюс IIoT для мониторинга; для модернизации — внедрение IIoT через OPC-шлюзы без замены оборудования; для новых «умных» производств — архитектура Индустрии 4.0 с киберфизическими системами и цифровыми двойниками. Ключевые компоненты и технологии цифровых двойников представлены на рис. 3 и рис. 4.

Для создания цифровых двойников требуются сбор данных, мониторинг, прогнозирование и моделирование.

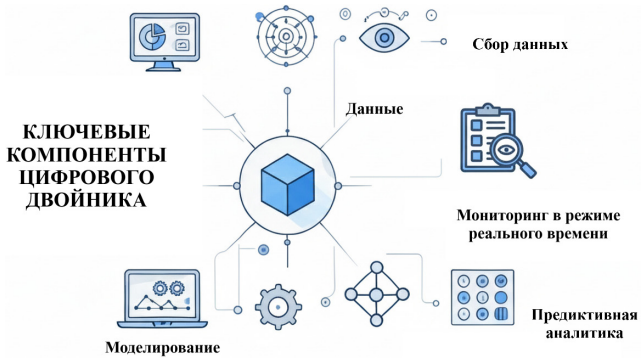

Рис. 3. Ключевые компоненты цифровых двойников

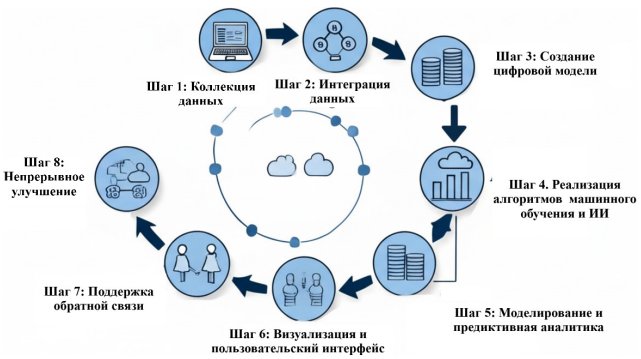

Рис. 4. Создание цифровых двойников

Процесс автоматизации и цифровизации включает установку датчиков, объединение данных на IoT-платформе, создание виртуальной копии, анализ с помощью ИИ, работу интерфейса для отслеживания показателей и обмен информацией между объектами и пользователями цифровой системы промышленного комплекса.

Заключение

Не существует универсального «лучшего» решения. Эволюция технологий автоматизации движется от жестких, иерархических систем (DCS) к гибким, открытым и сетевым (IIoT). Будущее за гибридными подходами, где надежность DCS на уровне управления процессом сочетается с аналитической мощью и гибкостью IIoT-платформ на уровне управления предприятием. Ключевым фактором успеха становится не столько выбор конкретной платформы, сколько способность предприятия выстроить сквозную цифровую стратегию, обеспечивающую свободный поток данных и их преобразование в реальные бизнес-ценности.

Литература

1. Next Move Strategy Consulting. Отчет о рынках распределенных систем управления / Next Move Strategy Consulting. – Текст : электронный. – URL: https://www.researchnester.com/ru/reports/distributed-control-system-market/4802 (дата обращения: 20.11.2025).
2. International Federation of Robotics (IFR). World Robotics 2025 report / IFR Statistical Department. – Frankfurt am Main : VDMA Services GmbH, 2025. – Текст : электронный. – URL: https://ifr.org/worldrobotics (дата обращения: 24.10.2025).
3. Russia Data Analytics Market to Surge at 20.96% CAGR, Reaching USD 13.96 Billion by 2032 / Infinium Global Research. – Текст : электронный. – URL: https://www.openpr.com/news/4242726/russia-data-analytics-market-to-surge-at-20-96-cagr-reaching (дата обращения: 01.11.2025).
4. Рылов С. Распределенные системы управления (РСУ)-DCS / С. Рылов. – Текст : электронный // Школа Fine Start : [сайт]. – URL: https://finestart.school/media/Distributed_control_systems (дата обращения: 20.11.2025).
5. Вектор ТМХ : журнал для партнеров АО «Трансмашхолдинг» / главный редактор К. Н. Дорохин. – 2023. – № 3. – 24 с. : ил. – ISSN 2949-5261. – Текст : электронный. – URL: https://rollingstockworld.ru/wp-content/uploads/2023/10/transmashholding_03_2023.pdf (дата обращения: 24.10.2025).
6. РУСЭНЕРГОСБЫТ : официальный сайт компании. – Текст : электронный. – URL: https://ruses.ru/ru/clients/service/ (дата обращения: 24.10.2025).
7. Система автоматизированная информационно-измерительная коммерческого учета электроэнергии (АИИС КУЭ) ООО «РУСЭНЕРГОСБЫТ» для энергоснабжения ОАО «РЖД». – Текст : электронный. – URL: https://all-pribors.ru/opisanie/95547-25 (дата обращения: 24.11.2025).
8. Системы мониторинга и диагностики состояния оборудования электрических сетей / НИИ «Энергосетьпроект». – Текст : электронный. – URL: http://ispu.ru/files/1._Otzyv_VO_Energosetproekt.pdf (дата обращения: 29.11.2025).
9. *Кнырвило В. Ю.* Автоматизированные системы управления для машиностроительных производств : учебное пособие / В. Ю. Кнырвило, А. В. Кнырвило, А. А. Погонин [и др.] ; под общей редакцией В. Ю. Кнырвило. – Санкт-Петербург : Лань, 2023. – 324 с. : ил. – ISBN 978-5-507-46123-7.
10. *Смирнов А. С.* Модели и методы интеграции MES- и SCADA-систем в единое информационное пространство предприятия / А. С. Смирнов, П. Д. Козлов, М. В. Новикова // Промышленные АСУ и контроллеры. – 2020. – № 8. – С. 34–42.

11. *Вольф Е. Р.* Промышленный Интернет вещей (IIoT): вызовы и решения для распределенных систем управления / Е. Р. Вольф, Ж. К. Дюпон // Труды Международной научно-технической конференции «Автоматизация и информационные технологии» (Сочи, 15–18 окт. 2022 г.) / редкол.: И. И. Тихонов [и др.]. – Сочи : СГУ, 2022. – С. 112–118.

12. *Иванов Д. А.* Индустрия 4.0: технологии и их применение / Д. А. Иванов, С. Б. Петров, К. Л. Сидоров [и др.] // Автоматизация и производство. – 2021. – № 5. – С. 12–19.

13. Цифровизация в производстве: шаги к Индустрии 4.0 // ООО «ПАИР». – Текст : электронный. – URL: https://pair-spb.ru/statyi/czifrovizacziya-v-proizvodstve-shagi-k-industrii-4-0 (дата обращения: 20.10.2025).

14. Гостев В. И. Цифровые двойники в промышленности: от концепции к внедрению / В. И. Гостев, Н. Н. Гусев. – Москва : ИНФРА-М, 2022. – 215 с.

УДК 519.67

ПРОБЛЕМА ТОЧНОСТИ МАТЕМАТИЧЕСКИХ ВЫЧИСЛЕНИЙ В PYTHON И МЕТОДЫ ЕЁ ПОВЫШЕНИЯ НА ПРИМЕРЕ ФУНКЦИЙ БЕССЕЛЯ

О. Ю. Лопухинский, Л. А. Минин

Воронежский государственный университет

Аннотация. В работе исследуется проблема точности вычисления функций Бесселя на языке Python и методы её повышения с учётом оптимизации быстродействия. Проведено сравнительное тестирование встроенных средств языка, а также библиотек Decimal, Fractions, Mpmath и SciPy при расчёте функции Бесселя первого рода нулевого порядка через ряд Тейлора. Показано, что библиотека Mpmath демонстрирует наилучшее сочетание высокой точности и производительности, в то время как SciPy обеспечивает достаточную точность для большинства практических задач при высокой скорости вычислений. Определены рекомендации по выбору инструментария в зависимости от требований к точности и производительности.

Ключевые слова: функции Бесселя, точность вычислений, Python, ряды Тейлора, Mpmath, SciPy, Decimal, Fractions, производительность, численные методы, сравнение библиотек.

Введение

Точность вычислений и производительность являются ключевыми требованиями при реализации математических моделей в программном обеспечении, особенно в таких областях, как оптика, обработка сигналов и управление сложными системами. Язык Python, несмотря на простоту использования, по умолчанию обеспечивает ограниченную точность вычислений. Однако есть способы повышения точности вычислений, что требует применения специализированных библиотек.

В качестве примера рассматриваются функции Бесселя — важный математический аппарат, широко используемый в науке и технике. В статье анализируются методы повышения точности их расчёта в Python с использованием библиотек Decimal, Fractions, Mpmath и SciPy, а также оценивается влияние выбранного метода на быстродействие [3, 4, 5, 6].

1. Методы вычисления функций Бесселя

1.1. Ряд Тейлора для функций Бесселя

Функция Бесселя первого рода нулевого порядка может быть представлена в виде ряда:

$$J_0(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(k!)^2} \left(\frac{x}{2} \right)^{2k}.$$

Данный ряд сходится для всех вещественных значений x, однако скорость сходимости и точность вычисления существенно зависят от используемого численного метода [1].

1.2. Библиотеки для вычислений с повышенной точностью

Decimal — библиотека для десятичной арифметики с фиксированной точностью и пользовательским управлением округлением [5].

Fractions — модуль для работы с рациональными числами, представляя их в виде рациональных дробей. Обеспечивает достаточно точные вычисления без потерь, но отличается низкой производительностью и подходит для задач, требующих абсолютной точности [6].

Mpmath — специализированная библиотека для вычислений с задаваемой точностью, включающая реализации специальных функций. Оптимизирована для математических расчётов высокой точности [4].

SciPy — библиотека для научных вычислений, использующая оптимизированные алгоритмы и стандартную арифметику с плавающей точкой. Обеспечивает высокую производительность и достаточную точность для большинства прикладных задач [3].

2. Экспериментальное сравнение точности

2.1. Методика эксперимента

Для сравнения точности различных библиотек было выполнено вычисление функции $J_0(0.5)$ с использованием каждого из рассматриваемых подходов. В качестве эталонного значения использовался результат библиотеки Mpmath [4] с установленной точностью 100 десятичных знаков.

2.2. Программная реализация

Для сравнительного анализа точности вычисления функций Бесселя была разработана программа на языке Python с использованием следующих библиотек: math, decimal [5], fractions [6], mpmath [4], scipy [3].

Для каждого метода была реализована функция вычисления $J_0(0.5)$ через разложение в ряд Тейлора:

Реализация с использованием встроенных float:

```
def bessel_builtin(x, n_terms=50):
    result = 0.0
    for k in range(n_terms):
        term = ((-1) ** k) * ((x / 2) ** (2 * k)) / (math.factorial(k) ** 2)
        result += term
        if abs(term) < 1e-20:  # Критерий остановки
            break
    return result
```

Реализация с использованием Decimal:

```
decimal.getcontext().prec = 50
def bessel_decimal(x, n_terms=100):
    x = decimal.Decimal(str(x))
    result = decimal.Decimal(0)
    for k in range(n_terms):
      term = (-1) ** k * (x/2) ** (2*k) / decimal.Decimal((math.factorial(k)) ** 2)
        result += term
    return result
```

Реализация с использованием Fractions:

```
def bessel_fractions(x, n_terms=20):
    x_frac = Fraction(x)
    result = Fraction(0)
    for k in range(n_terms):
        numerator = (-1) ** k * (x_frac / 2) ** (2 * k)
        denominator = Fraction(math.factorial(k)) ** 2
```

```
        term = numerator / denominator
        result += term
    return result
```

Реализация с использованием mpmath (для x=0.5):

```
mpmath.mp.dps = 100
x = 0.5
result = mpmath.besselj(0, x)
```

Реализация с использованием scipy (для x=0.5):

```
x = 0.5
result = float(scipy.special.jv(0, x))
```

2.3. Результаты сравнения

Результаты сравнения точности и скорости вычисления $J_0(0.5)$ приведены в табл. 1, табл. 2, табл. 3

Таблица 1

Точность и скорость встроенных методов и SciPy

Метод	Результат	Точность	Время
Встроенные функции	0.938469807240813	15 знаков после запятой	~0.00001 сек.
Scipy	0.93846980724081297	17 знаков после запятой	~0.00004 сек.

Таблица 2

Точность и скорость метода Fractions

Результат	Точность	Время
$499655 \cdot 10^{50} / 532414 \cdot 10^{50}$	60 знаков после запятой	~0.0002 сек.

Таблица 3

Скорость методов Mpmath и Decimal (точность 100 знаков)

Метод	Последние 8 знаков после запятой из 100	Время
Mpmath	93553603	~0.0000525 сек.
Decimal	93553600	~0.000551 сек.

2.4. Анализ результатов сравнительного тестирования

Проведенное исследование позволяет оценить сильные и слабые стороны каждого из рассмотренных инструментов для вычисления функций Бесселя.

• Встроенные функции Python (float) показывают наивысшую скорость, что делает их пригодными для задач, где производительность критична, а требования к точности умеренны (например, прототипирование, визуализация, образовательные проекты). Однако точность ограничена точностью стандартной арифметики с плавающей запятой.

• Библиотека SciPy [3] демонстрирует отличное сочетание скорости и точности, предоставляя результат с точностью до 17 знаков после запятой. Это оптимальный выбор для большинства прикладных научных и инженерных задач, где важна высокая производительность при работе с большими массивами данных.

• Библиотека Fractions [6] обеспечивает теоретически бесконечную точность за счет работы с рациональными числами, что исключает ошибки округления. Однако этот метод обладает наихудшей производительностью и на практике применим лишь для вычислений с неболь-

шим количеством членов ряда или в символьных вычислениях, где точность представления дроби абсолютно критична.

• Библиотека Decimal [5] позволяет достигать заданной высокой точности (в эксперименте — 100 знаков), что делает её мощным инструментом для финансовых расчётов и задач, требующих контроля над округлением. Тем не менее, её производительность существенно ниже, чем у Mpmath и SciPy.

• Библиотека Mpmath [4] продемонстрировала наилучший баланс между высокой точностью (возможность задания произвольного количества знаков) и приемлемой производительностью. Это делает её предпочтительным выбором для задач, требующих гарантированной высокой точности: фундаментальные научные расчёты, криптография, верификация алгоритмов и точный численный анализ.

Выбор библиотеки представляет собой компромисс между точностью и быстродействием. Для высокопроизводительных вычислений с достаточной для приложений точностью оптимален SciPy [3]. Для вычислений с произвольной точностью и её полным контролем наилучшим выбором является Mpmath, в то время как Decimal служит надежной альтернативой для десятичной арифметики. Fractions и встроенные float занимают свои узкие, но важные ниши [4–6].

Заключение

В рамках данной работы была исследована проблема ограниченной точности математических вычислений в Python и методы её решения на примере расчёта функции Бесселя первого рода нулевого порядка. Несмотря на простоту использования, стандартные средства языка не всегда удовлетворяют требованиям высокоточных научных расчётов.

Для решения этой проблемы было проведено сравнительное тестирование различных подходов: от встроенной арифметики с плавающей точкой и рациональных дробей (Fractions [6]) до специализированных библиотек для вычислений с произвольной точностью (Decimal [5], Mpmath [4]) и высокопроизводительных научных инструментов (SciPy [3]). В качестве базового метода использовалось разложение функции в ряд Тейлора, что позволило единообразно оценить как точность, так и производительность каждого подхода.

Результаты эксперимента наглядно продемонстрировали фундаментальный компромисс между точностью и быстродействием:

• Быстродействующие методы (встроенные float, SciPy [3]) обеспечивают приемлемую точность для большинства прикладных задач.

• Специализированные библиотеки (Mpmath [4], Decimal [5]) позволяют достигать произвольно высокой точности, ценой увеличения вычислительных затрат.

Таким образом, подтвержден основной тезис введения: для преодоления ограничений стандартных вычислений в Python необходимо и целесообразно привлекать внешние библиотеки. Настоящее исследование предоставляет практические рекомендации по их выбору: Mpmath [4] является оптимальным решением для задач, требующих гарантированной высокой точности, в то время как SciPy [3] предпочтительнее для высокопроизводительных вычислений, где достаточна стандартная точность. Данный вывод позволяет разработчикам и исследователям осознанно выбирать инструментарий, основываясь на конкретных требованиях к точности и производительности их проектов.

Литература

1. *Абрамовиц М.* Справочник по специальным функциям с формулами, графиками и математическими таблицами / М. Абрамовиц, И. Стиган ; под ред. В. А. Диткина, Л. Н. Карамзиной. – Москва : Наука, 1979. – 832 с.

2. IEEE Standard for Floating-Point Arithmetic. IEEE Std 754-2019 (Revision of IEEE 754-2008). – 2019. – P. 1–84. – DOI: 10.1109/IEEESTD.2019.8766229.

3. SciPy Reference Guide [Электронный ресурс] / SciPy Community. – 2023. – URL: https://docs.scipy.org/doc/scipy/reference/special.html (дата обращения: 15.05.2024).

4. Mpmath documentation [Электронный ресурс] / Fredrik Johansson. – 2023. – URL: http://mpmath.org/doc/current/functions/bessel.html (дата обращения: 15.05.2024).

5. Python Decimal documentation [Электронный ресурс]. – Python Software Foundation. – 2023. – URL: https://docs.python.org/3/library/decimal.html (дата обращения: 15.05.2024).

6. Python Fractions documentation [Электронный ресурс]. – Python Software Foundation. – 2023. – URL: https://docs.python.org/3/library/fractions.html (дата обращения: 15.05.2024).

7. *Вирт Н.* Алгоритмы и структуры данных / Н. Вирт. – Москва : Мир, 1989. – 360 с.

8. *Оленёв С. М.* Численные методы в научных расчётах / С. М. Оленёв, А. А. Семёнов. – Санкт-Петербург : Лань, 2018. – 336 с.

9. *Пресс У.* Численные рецепты / У. Пресс, С. Техольский, У. Фла́ннери. – 3-е изд. – Москва : Бином, 2007. – 818 с.

УДК 004.386

ВНЕДРЕНИЕ НОВОЙ СТРУКТУРЫ ПРОГРАММНОГО КОМПЛЕКСА ГИБРИДНОГО ВЫЧИСЛИТЕЛЬНОГО КЛАСТЕРА ВОРОНЕЖСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

А. Г. Потапов

Воронежский государственный университет

Аннотация. Использование гибридного вычислительного кластера в образовательном процессе требует поддержания его программного обеспечения в актуальном состоянии. Наличие в программной архитектуре слабых мест может приводить к потере производительности и сбоям в системе, что негативно сказывается на скорости выполнения научных и образовательных задач. Проведение реорганизации программного комплекса вычислительного кластера Воронежского государственного университета призвано ликвидировать имеющиеся недостатки и получить как можно большую производительность при текущей архитектуре.
Ключевые слова: вычислительный кластер, образовательный процесс, гибридная архитектура, Xeon Phi, сетевая загрузка, TFTP, xCAT, паравиртуализация, Xen-гипервизор, LDAP, SLURM, Linux.

Введение

Установка и настройка операционной системы на крупных вычислительных комплексах требует особой внимательности и скрупулёзности, поскольку выход из строя одного из компонентов тесно взаимосвязанной системы (к примеру, драйвера высокоскоростного интерконнекта InfiniBand) может привести к возникновению труднонаходимых проблем на всём кластере. С другой стороны, наличие в его составе нескольких узлов с идентичной архитектурой делает возможным унификацию процесса установки и настройки ключевых программных компонентов, что, в свою очередь, существенно упрощает их администрирование.

На протяжении года рассматривались различные варианты организации программной структуры вычислительного кластера ВГУ [1]. Некоторые из предложенных стратегий были отклонены в силу несовместимости с аппаратными компонентами и больших издержек при их настройке и сопровождении, другие же показали свою жизнеспособность и были успешно реализованы на практике. В данной статье будут приведены наиболее удачные решения, показавшие свою эффективность.

Первая часть посвящена процессу установки и настройки операционной системы в целом на каждый из вычислительных и служебных узлов кластера, включая использование технологий сетевой загрузки и паравиртуализации. Во второй части будут рассмотрены некоторые изменения в структуре отдельных программных компонентов, таких как LDAP-сервер и очередь исполнения SLURM.

1. Установка операционной системы

1.1. Сетевая загрузка на вычислительных узлах

Первым и наиболее важным шагом в настройке вычислительных узлов кластера является выбор типа загрузки операционной системы. С одной стороны, можно выполнить установку на локальных SSD-дисках, физически размещённых на каждом из настраиваемых узлов. В этом случае максимально упрощается настройка одного конкретного узла, на который ста-

нет возможным установить операционную систему с той же лёгкостью, с какой эта задача выполняется на домашнем компьютере.

С другой стороны, в ходе последующей настройки типично «кластерных» компонентов, включающих системы коммуникации, распределённую файловую систему, централизованное управление учётными записями пользователей и систему очередей исполнения, становится очевидным неэффективность и времязатратность такого подхода, поскольку каждый из компонентов приходится настраивать отдельно на каждом узле. Более разумным вариантом станет рассмотрение сетевой загрузки.

Структура кластера ФКН ВГУ неоднородна. В его составе 10 вычислительных узлов, однако семь из них имеют в своём составе сопроцессоры Intel Xeon Phi, а другие три — графические сопроцессоры NVIDIA Tesla K80. Установка драйверов, позволяющих работать с этими компонентами, предполагает внесение изменений в ядро Linux [2], которые несовместимы друг с другом, что делает невозможным использование одного ядра двумя архитектурами сразу. Следовательно, при планировании сетевой загрузки необходимо будет создать два сетевых корня (/nfs/root-MIC и /nfs/root-GPU), каждому из которых TFTP-сервер [3] будет назначать своё ядро.

Для реализации было решено использовать систему упрощённого управления крупными кластерами xCAT (Extreme Cloud Administration Toolkit) [4]. С её помощью стало возможным выполнить базовую настройку операционной системы на каждом из вычислительных узлов с учётом особенностей архитектуры сопроцессоров, прикладывая минимум усилий по настройке отдельных компонентов, таких как DHCP, TFTP или NFS. В том числе, не было необходимости настраивать конфигурацию начального загрузочного окружения initrd, что является довольно трудоёмкой задачей при настройке сетевой загрузки вручную.

1.2. Виртуальные служебные узлы

Изначальная структура кластера (рис. 1) предполагала наличие только одного головного узла, ответственного за взаимодействие всех аппаратных и программных компонентов между собой. Однако подобная схема работы подразумевает чрезмерную нагрузку на единственный сервер, что может приводить к замедлению и снижению стабильности его работы.

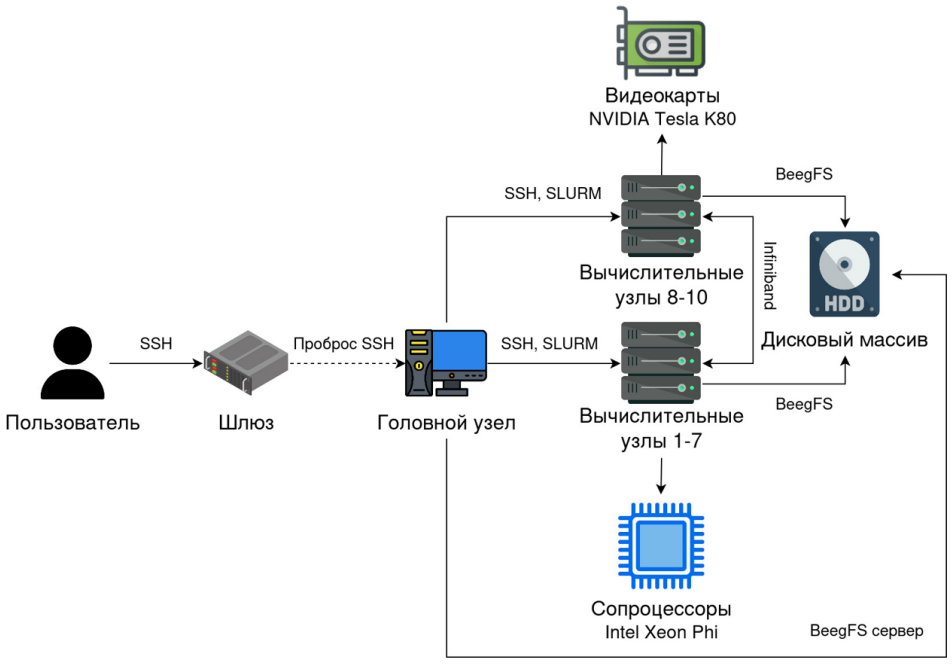

Рис. 1. Изначальная структура кластера

Для решения проблемы было решено воспользоваться ресурсами шлюза, функция которого ранее заключалась лишь в изоляции внутренней сети кластера от внешней сети ФКН. Благодаря загрузке в режиме Xen-гипервизора [5], подразумевающего адаптацию Linux-ядра для упрощения взаимодействия гостевой ОС с хостом, появилась возможность одновременно развернуть четыре виртуальных служебных узла без потери производительности. В результате задача бывшего головного узла сократилась до управления файловым хранилищем и содержания xCAT-сервера для сетевой загрузки.

Итоговая схема узлов вычислительного кластера представлена на рис. 2.

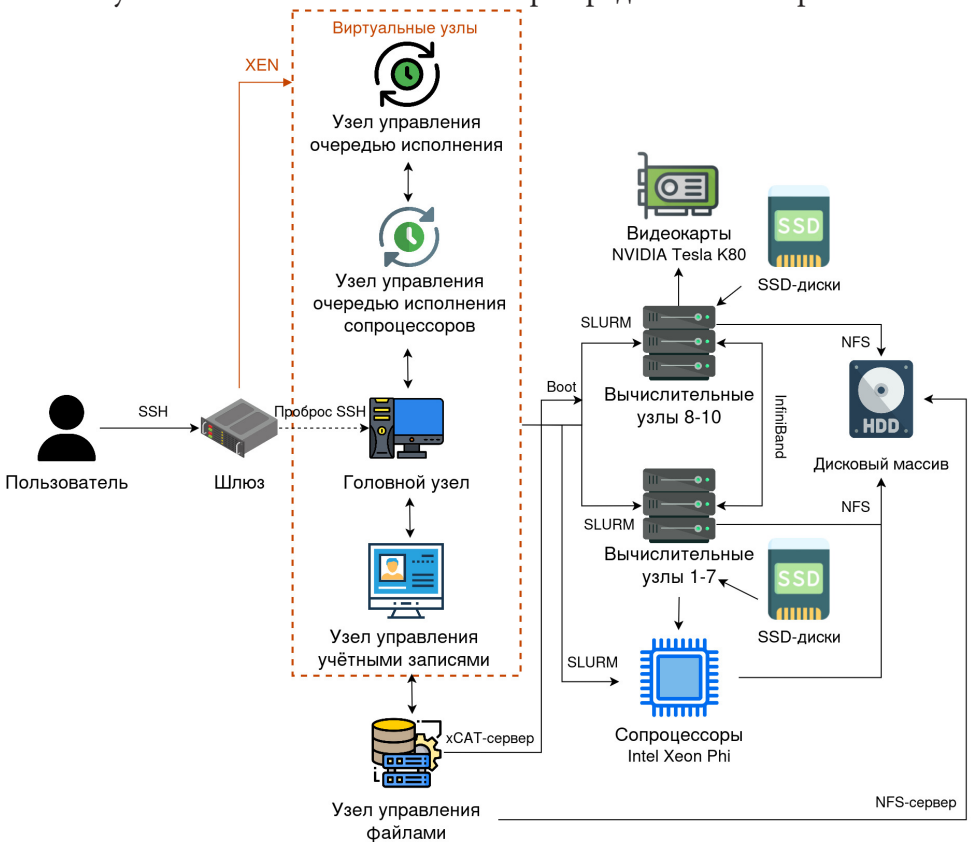

Рис. 2. Итоговая структура кластера

2. Изменение программного комплекса

2.1. Появление LDAP-сервера

Появление новых виртуальных узлов дало возможность более детально распределить нагрузку по кластеру. В частности, были внесены изменения в политику управления пользователями и группами. До появления LDAP-сервера данные об учётных записях хранились в стандартных расположениях Linux: /etc/passwd, /etc/group, /etc/shadow. И хотя такой способ чрезвычайно прост для настройки и понимания, он не обладает достаточной эффективностью для использования на кластерной системе, поскольку любое изменение необходимо дублировать на каждый из узлов.

Теперь же за хранение данных об учётных записях пользователей и группах отвечает отдельный узел, соединение с которым возможно даже на сопроцессорах Xeon Phi, что позволяет осуществлять быстрый и централизованный контроль, не нагружая при этом другие узлы, включая головной. Кроме того, при помощи LDAP возможно хранение дополнительной информации о пользователях, что также может оказаться полезным при работе с большим количеством студентов и преподавателей.

2.2. Создание двух очередей исполнения

Одним из ключевых преимуществ использования виртуальных машин стала возможность одновременного развёртывания двух серверов SLURM, отвечающих за организацию очереди исполнения программ на разнотипных сегментах кластера. Необходимость подобного решения обусловлена существенными различиями в архитектуре вычислительных узлов и сопроцессоров Xeon Phi.

Благодаря созданию сетевого подключения вида “external bridge” удалось установить связь каждого из четырнадцати сопроцессоров со всем кластером целиком, а не только с вычислительными узлами, к которым Xeon Phi присоединены физически. На всех было установлено дополнительное ПО, которое позволило получить к ним доступ в так называемом native-режиме, при котором не требуется участие хостового узла для проведения расчётов.

Однако устаревшая версия Linux-ядра 2.6 на сопроцессорах оказалась несовместимой с используемой на кластере версией SLURM, что привело к необходимости организации второго сервера специально для них, переключиться на который пользователи могут при помощи специально созданного модуля intel/2017, одновременно предоставляющего доступ к компилятору icc, поддерживающему ключ компиляции –mmic. Следует заметить, что обновление операционной системы на сопроцессорах также является невозможным ввиду прекращения их официальной поддержки с 2018 года.

Конечная структура очередей исполнения представлена на рис. 3.

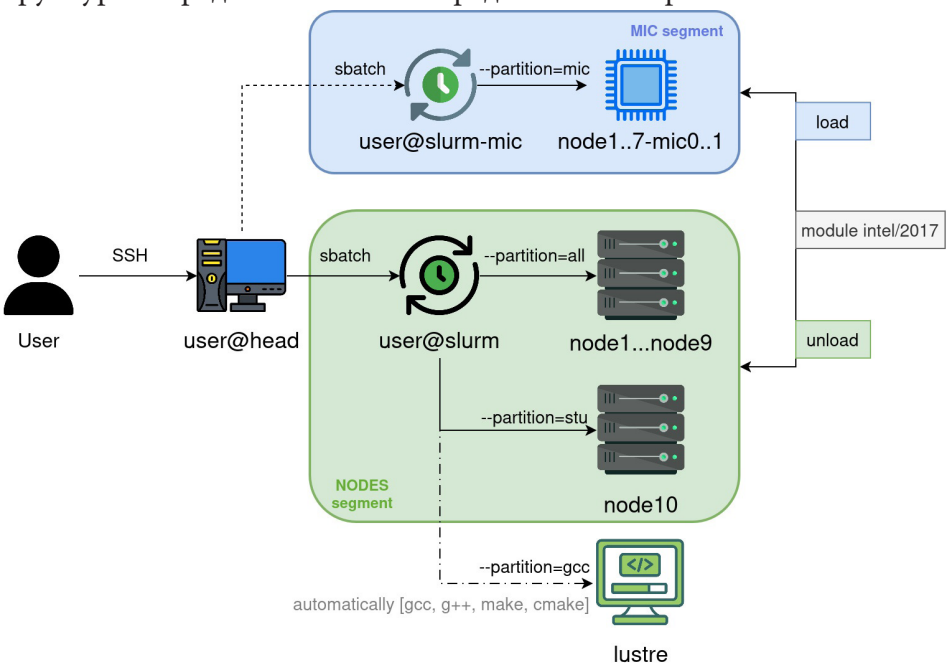

Рис. 3. Структура очередей исполнения на кластере

Заключение

В результате проведения реорганизации гибридный вычислительный кластер Воронежского государственного университета стал обладать более производительной и оптимально настроенной программной архитектурой, что существенно расширило возможности для обучения студентов навыкам параллельного программирования и написания выпускных квалификационных работ. Внесены изменения в логику загрузки вычислительных узлов по сети, внедрено использование технологии паравиртуализации для создания нескольких служебных узлов. Имеющиеся на кластере сопроцессоры были переведены в native-режим, что позволи-

ло подключить каждый из них к централизованной очереди исполнения и, в конечном счёте, увеличить совокупную производительность вычислительного кластера.

Литература

1. *Потапов А. Г.* Новый подход к организации программной архитектуры гибридного вычислительного кластера ФКН ВГУ / А. Г. Потапов, А. В. Романов // Труды молодых учёных факультета компьютерных наук ВГУ. – Воронеж, 2025. – № 5. – С. 581–585.

2. The Linux Kernel Archives. – URL: https://www.kernel.org (дата обращения: 21.11.2025).

3. Упрощенный протокол передачи файлов (TFTP) // IBM. Документация : [сайт]. – URL: https://www.ibm.com/docs/ru/aix/7.1.0?topic=protocols-trivial-file-transfer-protocol (дата обращения: 21.11.2025).

4. Extreme Cloud Administration Toolkit. – URL: https://xcat-docs.readthedocs.io/en/stable/ (дата обращения: 21.11.2025).

5. Xen project. – URL: https://xenproject.org (дата обращения: 21.11.2025).