1. Цель проекта

Изучение, освоение, закрепление навыков владения современными промышленными методами и технологиями разработки программного обеспечения (ПО), в частности разработки ПО инструментального типа с последующим применением его для решения конкретных специализированных задач. Изучение эффективных методов численного решения математических задач большой размерности.

1. Цель разработки

Ожидаемым выходом является система распределения вычислений с возможностями автоматической горизонтальной масштабируемости. Эта система должна представлять собой подключаемую библиотеку, или несколько библиотек, и предоставлять инфраструктуру и каркасные средства для описания вычислительных заданий, их структуры с учетом декомпозиции на элементарные составляющие, а также методов их обработки. На основе такой системы, предполагается разработка распределенной среды для численного решения задач линейной алгебры/математического анализа – в виде подключаемой библиотеки, обеспечивающей инфраструктурные и каркасные средства, с оснащающим пользовательским интерфейсом.

1. Основные средства реализации

Технологическая основа для выполнения проекта обеспечивается платформой Java версии 7. В качестве технологии межпроцессного взаимодействия предполагается использовать RMI. Пользовательский интерфейс предполагается реализовать в виде веб-приложения.

Необходимо решить следующие вопросы:

1. Какова будет технология реализации пользовательского интерфейса? Примерные варианты:
	1. JSP
	2. Spring
	3. Javascript + html + css (Dojo, ExtJS)
	4. *ну нафик веб, пишем гуй*

В качестве инструментальных средств разработки предполагается использовать IntelliJ Idea/NetBeans/Eclipse, для обеспечения совместимости форматов проектов и build-процесса – maven; Dia – для подготовки UML и других диаграмм. В качестве системы контроля версий – Mercurial (либо git). Для Unit-тестирования предполагается использовать JUnit, для создания «заглушек» – mock’ов – mockito. Предполагается использовать центральный репозиторий и issue-tracking, обеспечиваемые, предварительно, Bitbucket.

Необходимо выбрать:

1. Какая система контроля версий:
	1. Mercurial
	2. git
2. Методология разработки

Разработка должна осуществляться по гибкой схеме – итерационно, причем на каждой итерации должна фиксироваться некоторая работающая версия системы. Для обеспечения работоспособности каждой версии, необходимо выполнять модульное тестирование и непрерывную интеграцию. Каждая итерация, таким образом, включает: проектирование, реализацию, тестирование, интеграцию; в этот процесс вовлекаются все разработчики. Обязательным является unit-тестирование. Допускается применение техники парного программирования.

1. Словарь терминов

|  |  |
| --- | --- |
| **Термин** | **Значение** |
| (Вычислительное) задание | Совокупность, включающая описание[[1]](#footnote-1) задачи и исходные данные для нее. |
| Вычислительная сеть | Система связанных элементов, обеспечивающая полное решение логически замкнутых задач. |
| Вычислительный узел | Элемент вычислительной сети, обеспечивающий выполнение атомарных вычислительных заданий |
|  |  |
|  |  |
|  |  |

1. Основные требования.
2. Требования к обеспечению распределения вычислений

Логически система распределения вычислений должна представлять собой *вычислительную сеть*, состоящую из, возможно, разнотипных узлов. Каждый узел представляет собой вычислительный ресурс, а связи между узлами отражают (функциональные) зависимости. Необходимо обеспечить представление в сети разделяемых ресурсов, используемых несколькими (в частности, всеми) вычислительными узлами.

* 1. Должны предоставляться средства описания вычислительных задач различного типа (структур входных и выходных данных)
	2. Должны предоставляться средства описания и декомпозиции на подзадачи процесса решения для задачи каждого типа
	3. Должна обеспечиваться автоматическая, либо полуавтоматическая (на основе заданной конфигурации) диспетчеризация выполнения задания – определение множества необходимых вычислительных узлов и распределение между ними подзаданий
	4. Должна быть обеспечена возможность конфигурирования вычислительной сети
1. Требования к численному решению задач линейной алгебры/математического анализа
	1. Среда должна обеспечивать численное решение задач:
		1. Линейной алгебры:
			1. СЛАУ – прямым(и) и итерационным(и) методам(и)
			2. *\*Проблемы собственных значений[[2]](#footnote-2)*
		2. Теории линейных ОДУ:
			1. решение (системы) линейных ОДУ первого порядка
			2. *\*\*\*решение краевых задач*
		3. *Математического анализа*:
			1. *\*Интерполяции функции (сплайнами)*
			2. *\*\*Численное интегрирование/дифференцирование*
	2. Среда должна обеспечивать перечисление поддерживаемых типов решаемых задач и доступных методов их решения (на уровне API)
	3. Для каждого поддерживаемого типа задач, система должна предоставлять возможность явного указания метода ее решения (если он поддерживается). *\*Среда должна предоставлять пользователю-программисту возможность реализации и использования собственных типов задач и собственных методов их решения*.
	4. Среда должна обеспечивать возможность автоматического выбора наилучшего метода решения для конкретной задачи, если поддерживается более одного метода решения задач данного типа.
	5. Среда должна предоставлять такие математические примитивы, как матрицы и векторы, и операции над ними:
		1. Арифметические
		2. Различные нормы
		3. Показатель обусловленности
		4. …
2. Требования к интерфейсу конечного пользователя
	1. Должна обеспечиваться возможность просмотра списка поддерживаемых типов решаемых задач и доступных методов их решения
	2. Должна обеспечиваться возможность ввода исходных данных:
		1. В автоматическом режиме, когда все значения генерируются системой (например, случайным образом) исходя из некоторых параметров (например, условия положительной определенности матрицы, показателя обусловленности матрицы и др.)
		2. В ручном режиме
		3. В смешанном режиме: автоматическая генерация системой, см. 3.2.1 с последующей коррекцией пользователем
		4. Должна предоставлять возможность предварительного просмотра пользователем всех исходных данных до начала процесса решения
		5. Для каждого поддерживаемого типа задач, должна предоставляться возможность явного указания метода ее решения (если он поддерживается)
	3. Система должна обеспечивать возможность просмотра пользователем всем выходных данных – результатов решения задачи
	4. Система должна предоставлять возможность проводить исследования эффективности отдельных методов:
		1. Должна обеспечиваться возможность замера количественных характеристик процесса решения, например, времени
		2. Для каждого решения, система должна предоставлять возможность просмотра статистической информации – значения измеренных характеристик в зависимости от размерности задачи/используемого метода
3. Требования к интерфейсу администратора
	1. Должна обеспечиваться возможность просмотра текущей конфигурации вычислительной сети
	2. Должна обеспечиваться возможность перестройки конфигурации вычислительной сети
4. Прочие требования
	1. Должна поддерживаться работа с сверхбольшими – до нескольких сотен мегабайт – объектами в качестве исходных и выходных данных.
5. Некоторые вопросы архитектуры.
* Должен ли каждый узел вычислительной сети обладать фиксированной ролью, т.е. выполнять только одну функцию, или будет допускаться динамической присвоение роли каждому узлу в зависимости от вычислительного задания?
* Должен ли быть процесс декомпозиции одноуровневым, когда входное задание системы сразу декомпозируется на элементарные составляющие, которые система диспетчеризации распределяет затем по доступным узлам, либо такая декомпозиция может быть поэтапной?
1. Пример декомпозиции вычислительной задачи

Простейший пример декомпозиции вычислительной задачи – вычисление произведения матрицы на число. Предположим, что матрица представима в виде системы векторов-строк. Тогда процесс вычисления можно разделить на несколько частей:

1. формирование системы отдельных векторов (представляющих собой строки исходной матрицы);
2. для каждого выделенного вектора, вычисление результирующего вектора-произведения;
3. формирования итоговой матрицы из отдельных полученных векторов-строк.

При этом, очевидно, второй шаг может быть разбит на некоторое количество подзадач, каждая из которых может решаться независимо от других (параллельно с ними). Например, множество исходных векторов может быть разбито на несколько попарно непересекающихся подмножеств, каждое из которых может быть в виде единого пакета направлено на обработку в некоторый вычислительный узел.

1. Отдельный вопрос – форма этого описания. Например, описание может быть в форме некоторого класса C#; исходные данные тогда могут быть в форме значений свойств этого класса. [↑](#footnote-ref-1)
2. По мере возможности. Чем меньше звездочек «\*», тем выше приоритет. [↑](#footnote-ref-2)