ИНФОРМАТИКА

2008 **июль-сентябрь** **№ 3**

УДК 004.722.25

Д.А. Стрикелев

РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ СО СЛОЖНОЙ ЦЕЛЕВОЙ ФУНКЦИЕЙ НА ОСНОВЕ ТЕХНОЛОГИИ JAVA RMI И БИБЛИОТЕКИ JGAP

Рассматривается библиотека для программирования генетических алгоритмов на языке Java, исследуются встроенные средства распараллеливания вычислений в гетерогенной сети. Предлагается альтернативный программный каркас для разработки параллельных генетических алгоритмов на основе технологии Java RMI, демонстрируется его эффективность для решения оптимизационных задач со сложной целевой функцией.

Введение

Увеличение масштаба и сложности математических моделей, применяемых для описания систем и процессов из различных отраслей науки и промышленности, обусловливает высокую актуальность исследований, направленных на разработку численных методов для оптимизации подобных систем. Одним из популярных подходов к решению данного класса задач является использование генетических алгоритмов (ГА) [1–3].

Одной из основных проблем, возникающих при проектировании систем с помощью ГА, можно назвать высокую вычислительную сложность целевой функции (ЦФ). Для решения этой проблемы успешно применяются методы распределения вычислений между узлами [4, 5], но известные разработки обладают рядом недостатков: специализированы для конкретного класса задач, сложны в конфигурировании и расширении, чаще всего не являются кроссплатформенными.

Данное исследование имеет своей целью преодоление отмеченных недостатков и посвящено разработке универсального программного каркаса для распределения вычислений на основе технологии Java RMI и его интеграции с библиотекой программирования ГА в среде Java – JGAP (Java Genetic Algorithm Package).

1. Общие принципы построения генетических алгоритмов

ГА являются поисковыми процедурами, основанными на механизмах естественного отбора и генетики. В процессе вычислений ГА оперируют не одним решением, а целым их множеством – *популяцией.* Каждое решение, называемое *представителем* (или *индивидом*), может быть любой комбинацией значений переменных задачи, заданных в структуре данных, которая известна также как *хромосома*. Мера оптимальности каждого из решений (*фитнес*) является скалярной величиной и вычисляется с помощью специально определенной ЦФ.

Хромосома состоит из последовательности *генов*, представляющих переменные задачи. Каждый ген может принимать значения только из фиксированного множества – *аллеля.*

В пространстве решений осуществляется поиск с целью нахождения глобального экстремума ЦФ. Процесс поиска, называемый также *эволюцией,* производится на основе выбранных правил (операторов): *селекторов* и *генетических операторов*. Селекторы предназначены для отбора части индивидов из популяции и могут применяться до и после генетических операторов. Наиболее известными среди селекторов являются оператор *бридинга* (от англ. breeding, «скрещивание») и оператор *выживания*. Бридинг, применяемый до генетических операторов, формирует из популяции наборы пар для скрещивания, а оператор выживания, применяемый после, отбирает из разросшейся в результате скрещивания популяции те индивиды, которые перейдут в следующее поколение. Генетические операторы предназначены для изменения текущей популяции и формирования потомков на основе генетического материала родителей. Можно отметить следующие генетические операторы: *кроссо-*

вер, обеспечивающий обмен фрагментами хромосом родителей; *мутация*, поддерживающая разнородность значений генов и удерживающая популяцию от досрочного схождения к локальному экстремуму; *инверсия* и *упорядочение*, изменяющие порядок следования генов в хромосоме.

Цикл базового ГА состоит из следующих этапов:

1. Сформировать начальную популяцию.
2. Пока соблюдается условие повторения, выполнить следующие шаги:
   - 2.1. Отобрать особей, подлежащих скрещиванию, и сформировать среди отобранных особей пары (оператор бридинга).
   - 2.2. Произвести скрещивание (оператор кроссовера):
     - 2.2.1. Случайным образом выбрать точки разрезания хромосом предков.
     - 2.2.2. Произвести обмен фрагментами хромосом предков.
   - 2.3. Произвести мутацию популяции (оператор мутации).
   - 2.4. Произвести контроль потомков на предмет соблюдения ограничений (оператор ремонта).
   - 2.5. Вычислить ЦФ для новых и измененных членов популяции.
   - 2.6. Произвести селекцию внутри разросшейся популяции (оператор выживания).
3. Завершить эволюцию.

В общем случае вычислительная сложность операторов бридинга и выживания зависит от размера популяции, операторов кроссовера и мутации – от длины хромосомы, оператора ремонта – от обоих параметров. Наиболее трудно прогнозируемой является вычислительная сложность ЦФ, так как в зависимости от задачи она может как линейно, так и нелинейно зависеть от длины хромосомы. Для снижения общего времени работы ГА и повышения качества получаемых решений эволюцию распараллеливают различными способами в зависимости от того, какой из этапов ГА потребляет основную часть времени.

2. Анализ способов распараллеливания генетических алгоритмов

По перечню операций, которые выполняются на каждом вычислительном узле, участвующем в распределенной эволюции, логике взаимодействия узлов и набору пересылаемых данных можно выделить такие виды параллельных ГА, как «главный-подчиненный», «расширенный главный-подчиненный», «подчиненные колонии» и «независимые нации». Из анализа их характеристик (табл. 1) следует:

– виды «подчиненные колонии» и «независимые нации» предназначены, главным образом, для более широкого покрытия области решений индивидами (т. е. повышения качества получаемых решений), в то время как две разновидности «главный-подчиненный» предназначены для ускорения процесса эволюции, проходящего на главном узле;

– вид «расширенный главный-подчиненный» является предпочтительным в том случае, если время, необходимое на проведение скрещивания и мутации двух индивидов, превышает время передачи индивидов подчиненным узлам.

Из полученных результатов видно, что наибольший выигрыш по общему времени эволюции ГА может быть достигнут при использовании первых двух видов параллельных алгоритмов, так как отдельные домены «подчиненных колоний» и «независимых наций» не увеличивают скорость вычисления ЦФ, тратя ресурсы на обработку собственных индивидов. Выбор же одного из видов «главный-подчиненный» или «расширенный главный-подчиненный» производится в зависимости от используемых операторов кроссовера и мутации: после их применения может потребоваться ремонт хромосом для получения практически реализуемых и удовлетворяющих ограничениям индивидов, что является в общем трудоемкой операцией.

3. Технологии реализации параллельных генетических алгоритмов

При рассмотрении потенциальных технологий реализации параллельных ГА в качестве основных критериев выбора были выделены следующие:

Таблица 1

Характеристики существующих видов параллельных ГА

Вид ГА	Описание	Данные, передаваемые на подчиненные узлы	Данные, передаваемые на главный узел	Вычисления в удаленном узле	Преимущества	Недостатки
«Главный-подчиненный»	Главный узел делегирует ряд вычислительных задач на подчиненные узлы. Существует только одна популяция на главном узле	Хромосома	Фитнес в виде числа с плавающей точкой	ЦФ	Легок в реализации, по сети передаётся малый объем данных	Низкий уровень параллельности, высокие затраты на синхронизацию, высокая нагрузка на главный узел
«Расширенный главный-подчиненный» (с генерацией индивидов на стороне подчиненных узлов)	Главный узел делегирует расширенный круг задач на подчиненные узлы, вместе с этим увеличивается объем передаваемых по сети данных	Хромосомы: одна для вычисления фитнеса и мутации, пара для скрещивания	Фитнес в виде числа с плавающей точкой, хромосома (после мутации), пара хромосом (после скрещивания)	ЦФ, мутация, скрещивание	Достаточно легок в реализации, большее количество вычислений делегируется на подчиненные узлы	По сети передается больший объем данных, затраты на синхронизацию могут быть велики
«Подчиненные колонии»	Каждый узел имеет свою автономную популяцию. После того как популяция создана, она пересылается обратно на главный узел. Главный узел объединяет все популяции подчиненных узлов в новое поколение, которое затем рассылается подчиненным для новой итерации	Популяция	Популяция (переменного размера)	Все операции, кроме объединения в итоговое поколение	Высокий уровень параллельности	По сети передается значительный объем данных
«Независимые нации»	Не существует главного узла (начальный узел только инициализирует параметры алгоритма и запускает вычисления). Каждый узел имеет собственную популяцию, которая развивается независимо от других. В конце каждой итерации узлы обмениваются индивидами, после чего начинается создание новой популяции	Переменное количество хромосом (в зависимости от параметров алгоритма)	–	Все операции	Максимальный уровень параллельности, большая распределенность кандидатов решений в пространстве решений, балансирование нагрузки всех узлов	Возможна более медленная сходимость к наилучшему решению

– легкость внесения изменений и модернизации разработанного кода;
– использование объектно-ориентированной парадигмы программирования;
– скорость и эффективность работы кода;
– наличие готовых к использованию, протестированных и апробированных разработок;
– кроссплатформенность и переносимость кода;
– наличие надежных и апробированных средств распределения работ в сети;
– наличие средств, позволяющих выполнять априорно неизвестный программный код на удаленных узлах, которые не должны обладать какими-либо сведениями об алгоритме, реализуемом в рамках оптимизационного процесса, но предоставлять ему в пользование свои локальные ресурсы процессора и оперативной памяти.

Среди возможных альтернатив рассматривались такие варианты, как платформа Java и библиотека JGAP, платформа .NET и версия JGAP для .NET, язык C++ с использованием Sockets API и библиотека GALib, разработка собственной библиотеки для ГА на интерпретируемом языке (PHP, Perl, TCL). Реализация собственной библиотеки в том случае, если аналогичная разработка уже существует, не рассматривалась. Результаты анализа программных технологий приведены в табл. 2.

Таблица 2

Анализ программных платформ реализации ГА

Критерий	Java / JGAP	.NET / JGAP.NET	C++ / Sockets / GALib	PHP/ Perl / TCL
Легкость изменений и модернизации	высокая	высокая	низкая	средняя
Объектно-ориентированная парадигма	есть	есть	есть	есть
Скорость и эффективность работы	средняя	средняя	высокая	низкая
Наличие готовых разработок	есть	есть	есть	отсутствует
Кроссплатформенность и переносимость	высокая	низкая	средняя	высокая
Встроенные средства распределения работ в сети	есть	есть	отсутствует	отсутствует
Встроенные средства для выполнения априорно неизвестного программного кода на удаленных узлах	есть	есть	отсутствует	отсутствует

В результате за основу было решено взять платформу Java SE и библиотеку JGAP [6].

В основе иерархии классов JGAP лежат следующие абстракции: Allele (множество значений, которые может принимать ген), Gene (связывает конкретное значение гена из аллеля с его позицией в хромосоме – локусом), Chromosome (множество генов, обладающее фитнесом), FitnessFunction (реализует ЦФ и вычисляет значение фитнеса по переданной в качестве параметра хромосоме), Population (представляет множество хромосом), Genotype (содержит популяцию хромосом, принадлежащих одному поколению).

Для преобразования популяции во время эволюции предусмотрены абстракции GeneticOperator и NaturalSelector. Общая схема эволюции, в частности количество поколений, число хромосом в популяции, количество и тип генов в хромосоме, используемые генетические операторы и схема выживания и т. п., задается посредством объекта Configuration. В языке Java эти абстракции представлены интерфейсами, задающими перечень методов, которые должны поддерживаться каждой из конкретных реализаций объектов.

Для реализации ГА на основе JGAP необходимо выполнить следующие шаги:

1. Спланировать хромосому, выбрать ее размер и определить составляющие гены. Создать шаблонную хромосому, по примеру которой будут создаваться новые индивиды.

2. Реализовать класс, вычисляющий ЦФ.

3. Создать и настроить объект Configuration, связать с ним шаблонную хромосому, задать параметры популяции и эволюции, используемые генетические операторы и селекторы.

4. Создать объект Genotype, а с его помощью и популяцию. При этом в качестве шаблона для создания членов популяции будет использоваться шаблонная хромосома, созданная при выполнении шага 1.

5. Запустить эволюцию.

Действия, выполняемые на каждой итерации процесса эволюции, схожи с шагами этапа 2 базового ГА, приведенного в разд. 1, но их перечень отличается:

1. Определить наиболее приспособленного индивида с целью его предохранения от случайной потери в ходе итерации.
2. Отсеять часть индивидов для удержания размера популяции на постоянном уровне.
3. Вычислить ЦФ для новых или измененных индивидов.
4. Просеять популяцию с помощью селекторов.
5. Применить заданные генетические операторы.
6. Просеять популяцию с помощью селекторов.
7. Вычислить групповую ЦФ, значение которой зависит не только от ЦФ отдельных индивидов, но и от количества схожих индивидов в популяции.
8. Дополнить популяцию новыми случайными индивидами, если в ходе селекции ее размер опустился ниже минимально допустимого уровня.

4. Распараллеливание эволюции с помощью встроенных средств JGAP

Библиотека JGAP обладает встроенным механизмом распределения вычислений, осуществляя взаимодействие посредством сокетов (разработчиками данный механизм называется GRID JGAP). Анализируя взаимодействия между узлами, можно выделить следующие роли: «рабочий», «мастер» и «клиент».

Большинство узлов выполняют роль рабочего, получая от мастера запросы, требующие обработки. Каждый запрос содержит в себе экземпляр объектов Configuration, Genotype, Population и набор хромосом. Дополнительно с мастера передаются указания об алгоритмах (также называемых *стратегиями*) инициализации, эволюции и возврата, в соответствии с которыми рабочий действует. Стратегия инициализации используется для создания на основе переданной в запросе популяции исходной популяции, стратегия эволюции – для задания правил изменения популяции, стратегия возврата – для определения набора данных, которые рабочий должен вернуть мастеру (например, только значения ЦФ).

Роль мастера заключается в хранении списка всех подчиненных ему рабочих, распределении запросов, получаемых от клиента, а также возврате полученных от рабочих результатов клиенту.

Задачами клиента являются: разделение всей решаемой задачи на фрагменты, создание на их основе запросов, передача запросов мастеру, получение от него результатов, а также последующая их обработка.

Подобная схема вычислений позволяет с минимумом усилий реализовать собственную распределенную эволюцию ГА. Для этого необходимо лишь реализовать в виде отдельных классов целевую функцию, стратегии для рабочих и клиента, создать объект конфигурации и запустить вычисления.

Тем не менее, эксперименты с GRID-JGAP выявили ряд существенных недостатков, которые делают его малоприменимым для оптимизации компьютерных сетей:

1. Схема передачи хромосом на удаленные узлы, при которой каждая хромосома помещается в популяцию из одного индивида, популяция облекается в генотип, а генотип вместе с конфигурацией помещаются в запрос, который и передается рабочему, приводит к существенному перерасходу времени на передачу дополнительных объектов. Так как упаковка состояния объектов в поток байтов (*сериализация*) и восстановление состояния из него (*десериализация*) выполняются на одном узле, это становится узким местом в производительности системы.
2. Взаимодействие на основе сокетов предполагает, что для запуска рабочих необходимо наличие ожидающего подключений мастера, т. е. узлы должны запускаться в строго определенной последовательности.
3. Взаимодействие на основе сокетов приводит к тому, что прекращение работы одного из рабочих или сервера вызывает аварийное завершение работы всех остальных клиентов, после чего требуется их перезапуск.

4. На данный момент в GRID JGAP не предусмотрена возможность загрузки объектного кода классов на удаленные узлы автоматически, он должен там присутствовать до начала эксперимента.

Для устранения отмеченных недостатков было принято:

– разработать программный каркас на основе технологии удаленного вызова методов Java RMI (Remote Method Invocation), обеспечивающий независимость клиентской и серверной частей, автоматическую загрузку кода рабочими с мастера;

– интегрировать разработанный каркас с библиотекой JGAP;

– провести оптимизацию процедур обмена данными между вычислительными узлами.

5. Разработка программного каркаса для параллельных вычислений на основе технологии Java RMI и библиотеки JGAP

Разработать программный каркас для распределенных вычислений было предложено на основе технологии удаленного вызова методов Java RMI. Как известно, она позволяет вызывать методы объектов, расположенных на удаленных узлах [7].

Базовыми сущностями программного каркаса являются такие классы, как RemoteTask (УдаленнаяЗадача), RemoteTaskResult (РезультатУдаленной-Задачи), RemoteTaskPerformer (ИсполнительУдаленнойЗадачи), RemoteTaskWorker (РабочийУдаленнойЗадачи) и RemoteTaskScheduler (ПланировщикУдаленныхЗадач).

Класс RemoteTask инкапсулирует данные задачи, подлежащие параллельной обработке, и логику их обработки. Весь процесс решения состоит из двух частей: вычислений, производимых на удаленном узле (метод fulfil), и обработки результата удаленных вычислений (метод handleResult). Связующим звеном между двумя частями является объект класса RemoteTaskResult, хранящий сведения о результате удаленной операции.

Класс RemoteTaskPerformer представляет объекты, находящиеся на удаленных узлах, к методам которых, в данном случае к методу doRemoteOperation, происходит обращение с главного узла. Параметром, передаваемым в doRemoteOperation, является экземпляр класса RemoteTask, а возвращаемое значение имеет тип RemoteTaskResult. Параметр, передаваемый в метод, и возвращаемый результат для передачи проходят через сериализацию и десериализацию, а если на удаленном узле будет отсутствовать объектный код для необходимого класса, то его загрузка с главного узла будет произведена автоматически средствами Java RMI.

Класс RemoteTaskWorker является производным от класса потока выполнения (Thread) и предназначен для взаимодействия с объектом класса RemoteTaskPerformer. В ожидании завершения вычислений на удаленном узле RemoteTaskWorker блокируется, а после пробуждения выполняет обработку возвращенного результата и заново засыпает в ожидании обращений.

Объект класса RemoteTaskScheduler существует в единственном экземпляре и осуществляет постановку поступающих запросов в очередь ожидания, мониторинг состояния удаленных рабочих и извлечение запросов из очереди ожидания и закрепление их за рабочими при освобождении одного из них. В том случае, когда все рабочие заняты вычислениями либо очередь ожидания пуста, имеющийся внутри RemoteTaskScheduler поток выполнения переходит в спящий режим. Взаимосвязи классов программного каркаса в нотации языка UML показаны на рис. 1.

Для использования предложенного программного каркаса при решении задач, требующих распределения вычислений, достаточно создать потомков классов RemoteTask и RemoteTaskResult и реализовать в них требуемую функциональность удаленной и локальной обработок.

В случае применения библиотеки JGAP для создания ГА оптимизации компьютерных сетей распараллеливаемой операцией является вычисление ЦФ индивидов. Последовательность действий (см. разд. 3), которые выполняются в JGAP на каждом цикле эволюции, реализуется классом GABreeder (от англ. «заводчик»). Каждая итерация начинается с вычисления ЦФ для новых или измененных членов популяции. Чтобы выделить членов популяции, для которых ЦФ еще не вычислена, JGAP применяет внутреннее кэширование фитнеса хромосом. Для но-

вых индивидов, полученных при инициализации популяции, скрещивании или мутации, фитнес устанавливается в минус единицу (недостижимая для ЦФ величина), а затем в начале каждой итерации проверяется, какие индивиды имеют отрицательное значение фитнеса, и ЦФ вычисляется только для них.

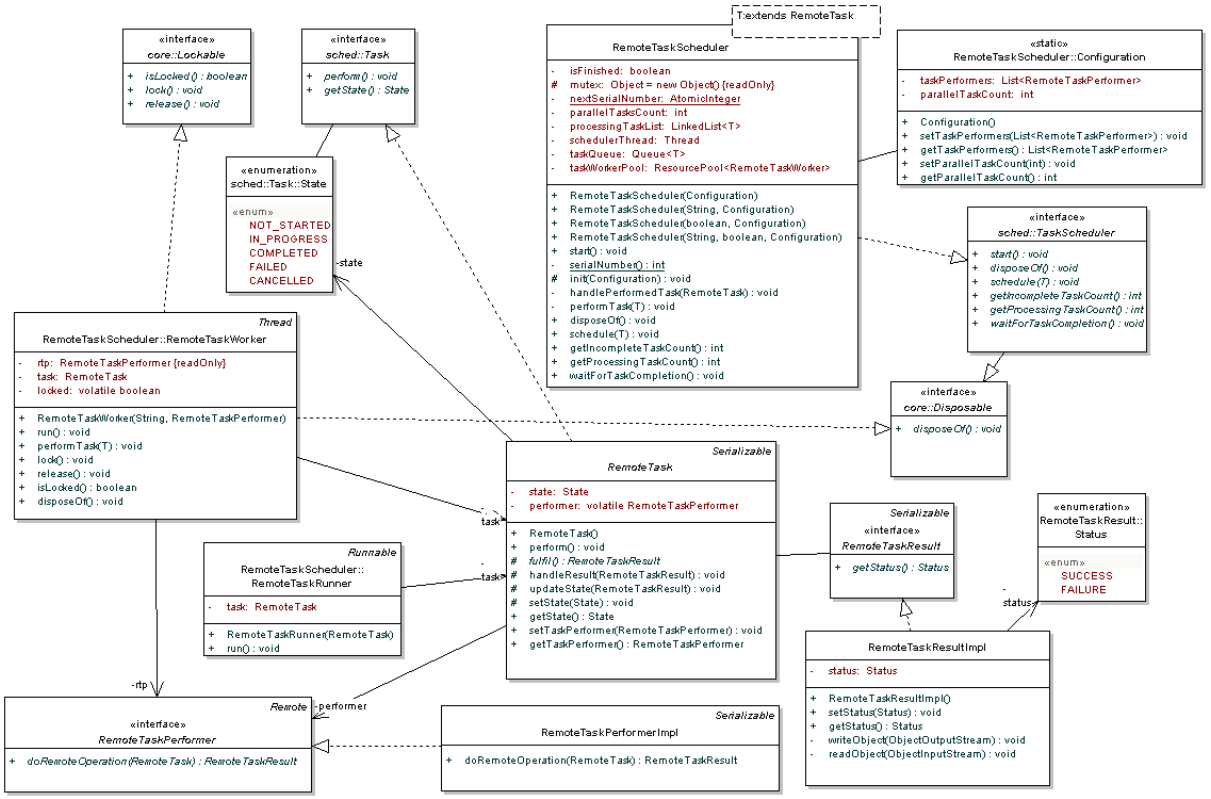

Рис. 1. Программный каркас распределенных вычислений на основе Java RMI

Указанная особенность позволяет создать класс DistributedGABreeder, который по отношению к базовому классу GABreeder является внешней оболочкой. Задача класса-оболочки заключается в параллельном вычислении ЦФ для всех новых индивидов популяции путем распределения запросов по узлам-рабочим. Таким образом, когда популяция передается для обработки внутреннему классу GABreeder, все индивиды обладают положительным значением фитнеса, и шаг 2.3 пропускается.

Для уменьшения временных затрат на сериализацию и десериализацию объектов при передаче заданий на удаленные узлы базовые версии классов Chromosome и Gene библиотеки JGAP были заменены новыми: DistributedChromosome и DistributedGene. Смысл произведенных изменений заключался в избавлении от необходимости сохранения/восстановления состояния ряда объектов, с которыми связано вычисление ЦФ хромосомы в JGAP: аллелей, популяции, генотипа, конфигурации ГА.

6. Вычислительный эксперимент

Для сравнения эффективности известного комплекса GRID JGAP и разработанного каркаса RMI JGAP была выбрана задача максимизации количества единичных генов в хромосоме при изменяющейся ее длине, варьируемой сложности ЦФ и различном числе узлов-вычислителей. В качестве самого простого способа изменения вычислительной сложности ЦФ было предложено поместить алгоритм вычисления ЦФ во внешний цикл, выполняемый N раз. Варьируя значение N, можно изменять общее время вычисления ЦФ.

В каждом из сравниваемых каркасов была реализована программа, представляющая одинаковый алгоритм с идентичным набором генетических операторов и селекторов, который ра-

ботает с популяциями одинакового размера и выполняет одинаковое количество итераций. Размер популяций был выбран равным 50, а количество поколений – 5. Небольшие значения параметров ГА объясняются тем, что предметом данного исследования являлось не сравнение качества работы алгоритмов, а эффективность распределения вычислений.

Эксперименты проводились в лаборатории, состоящей из 15 ЭВМ с процессорами AMD Athlon 64 X2 4400+ 2.3 ГГц и 2048 Мб оперативной памяти под управлением главного узла с процессором AMD Athlon 64 X2 4400+ 2.3 ГГц и 4096 Мб оперативной памяти. Все тесты были разбиты на три группы, для каждого теста проводилось по десять замеров и выполнялось усреднение. Необходимость этого обусловлена тем, что ГА являются вероятностными методами и время вычислений при каждом конкретном запуске может различаться. Кроме того, присутствующий в Java «сборщик мусора», ответственный за освобождение неиспользуемой памяти, может время от времени приостанавливать выполнение программ для освобождения памяти, а время и длительность таких пауз спрогнозировать невозможно.

Тест 1: измерялось ускорение, получаемое с помощью RMI JGAP при количестве узлов от 1 до 15, при длине хромосомы в 100, 500 и 1000 генов, задержке вычисления ЦФ в 5×10^3, $2{,}5\times10^5$, 5×10^5 циклов (табл. 3).

Таблица 3

Полученное ускорение при использовании RMI JGAP в экспериментах теста 1

Кол-во узлов	Задержка, циклов			Задержка, циклов			Задержка, циклов		
	5×10^3	$2{,}5\times10^5$	5×10^5	5×10^3	$2{,}5\times10^5$	5×10^5	5×10^3	$2{,}5\times10^5$	5×10^5
	Длина хромосомы – 100 генов			Длина хромосомы – 500 генов			Длина хромосомы – 1000 генов		
1	1,044	1,245	1,372	0,587	0,630	0,623	0,551	0,559	0,570
2	1,361	1,577	1,726	0,915	0,983	0,999	0,899	0,922	0,939
3	2,230	2,723	3,047	1,441	1,627	1,596	1,404	1,476	1,458
4	3,007	3,919	4,238	1,924	2,215	2,159	1,897	1,985	2,038
5	3,830	5,039	5,430	2,460	2,772	2,824	2,345	2,544	2,547
6	4,321	6,030	6,629	2,884	3,404	3,378	2,813	3,068	3,147
7	4,842	6,962	7,657	3,296	3,899	3,966	3,167	3,539	3,679
8	5,174	8,420	9,234	3,735	4,622	4,553	3,549	4,012	4,112
9	5,506	8,985	10,130	4,065	5,107	5,020	4,025	4,541	4,647
10	5,920	10,100	10,859	4,492	5,718	5,578	4,308	4,876	5,114
11	6,380	10,499	12,329	4,775	6,037	6,052	4,703	5,474	5,606
12	6,380	10,615	11,376	4,818	6,427	6,322	4,891	5,753	5,841
13	6,295	10,900	12,335	5,173	6,959	6,753	5,245	6,213	6,374
14	6,793	12,940	14,027	5,412	7,285	7,377	5,532	6,635	6,707
15	6,649	13,459	14,130	5,719	7,677	7,858	5,808	7,272	7,318
Время вычисления ЦФ, мс	4,5	145	289	32	1870	3899	96	5228	10783

Проанализировав полученные результаты, можно отметить, что при одном и том же числе узлов ускорение:

– увеличивается при увеличении сложности ЦФ, так как уменьшается время, уходящее на сериализацию и передачу объектов, по отношению к времени, уходящему на вычисления;

– уменьшается при увеличении размера хромосомы, так как увеличивается объем передаваемых данных. Для больших хромосом (от 500 генов) и простых ЦФ это приводит к тому, что при небольшом числе узлов-вычислителей (до трех) наблюдается не ускорение, а замедление вычислений, поскольку время, затраченное на сериализацию, превышает время, необходимое для завершения расчетов.

Тест 2: сравнивалось ускорение, получаемое с помощью RMI JGAP и GRID JGAP для количества узлов от 1 до 15, при длине хромосомы в 500 генов, задержке вычисления ЦФ в 5×10^3 и $2{,}5\times10^5$ циклов. Для задержки ЦФ в 5×10^3 циклов ускорение при GRID JGAP не смогло преодолеть порог в 0,16 даже на 15 узлах (рис. 2).

Тест 3: измерялось ускорение, получаемое с помощью RMI JGAP и GRID JGAP для 15 узлов, при длине хромосомы в 500 генов, задержке вычисления ЦФ в интервале от 0 до 5×10^5 циклов (рис. 3).

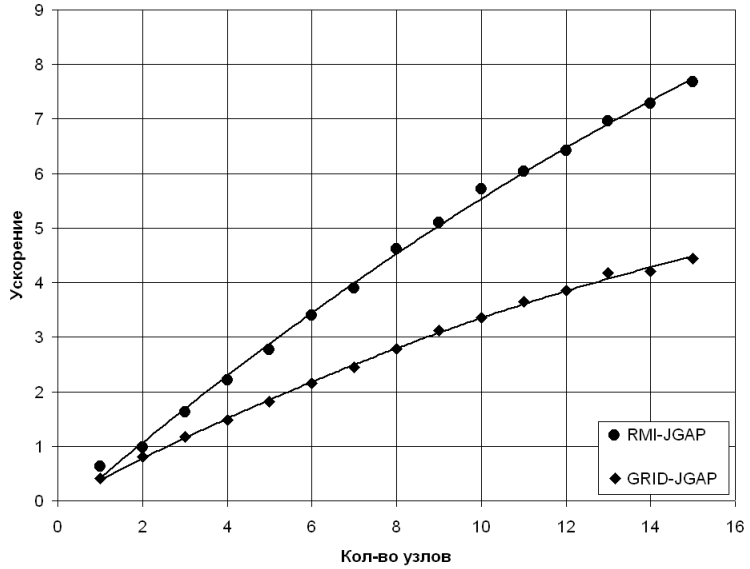

Рис. 2. Изменение ускорения при использовании RMI JGAP и GRID JGAP в экспериментах теста 2

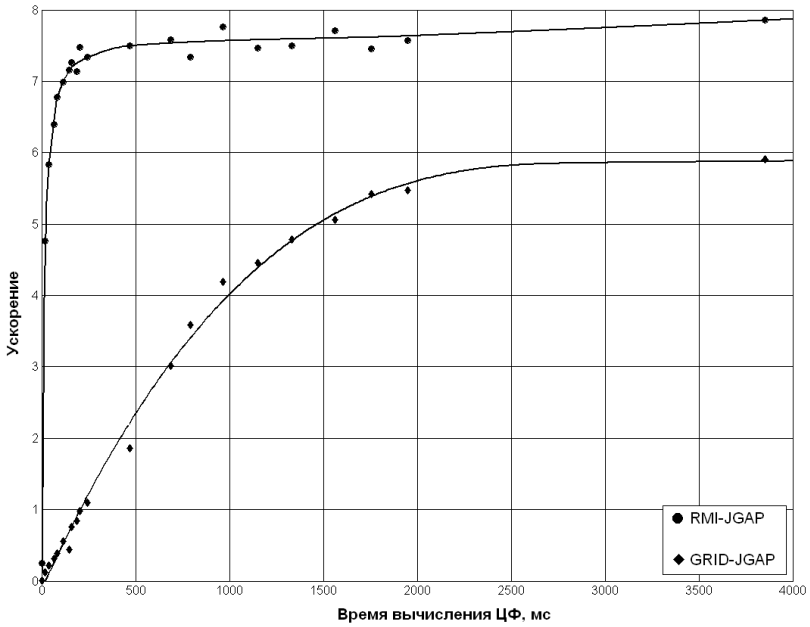

Рис. 3. Зависимость ускорения от времени вычисления ЦФ в экспериментах теста 3

Заключение

Разработанный программный каркас для распределенных вычислений на основе Java RMI и его интеграция с библиотекой JGAP позволяют преодолеть недостатки, присущие GRID JGAP: обеспечить произвольный порядок запуска распределенной вычислительной системы и возможность автоматической загрузки объектного кода необходимых классов с главного узла. Оптимизированная схема сериализации объектов позволяет получать выигрыш в производительности даже в случае ЦФ с временем вычисления до 500 мс. Для более сложных ЦФ RMI JGAP демонстрирует производительность, на 25–40 % большую по сравнению с GRID JGAP.

Комплекс RMI JGAP может эффективно применяться для решения оптимизационных задач со сложной ЦФ.

Список литературы

1. Воротницкий, Ю.И. О задаче размещения распределенного информационного сервера в интернете в условиях переменной нагрузки / Ю.И. Воротницкий, Д.А. Стрикелев // Управление информационными ресурсами: материалы Второй науч.-практ. конф., 16.03.2004 г. / Академия управления при Президенте Республики Беларусь. – Минск, 2004. – С. 44–46.

2. Habib, S.J. Automated design of hierarchical Intranets / S.J. Habib, A.C. Parker, D.C. Lee // Computer Communications. – 2002. – № 25. – P. 1066.

3. Topological Design of Survivable IP Networks Using Metaheuristic Approaches / E.C.G. Wille [et al.]. – Berlin: Springer, 2005. – 206 p.

4. Flores, S.D. Telecommunication Network Design with Parallel Multi-objective Evolutionary Algorithms / S.D. Flores, B.B. Cegla, D.B. Cáceres // LANC'03, October 3-5, 2003, La Paz. – Bolivia, 2003. – P. 1–11.

5. Goodman, E.D. An introduction to GALOPPS – the «Genetic ALgorithm Optimized for Portability and Parallelism» system [Computer file]: Technical Report 94-11-01 / Michigan State Univ. Computer data. East Lansing, 1994. – 1 CD-ROM.

6. Java Genetic Algorithm Package – JGAP / K. Meffert [et al.] // JGAP Website [Electronic Resource]. – Mode of access: http://jgap.sourceforge.net. – Date of access: 13.02.2008.

7. Дейтел, Х. Технологии программирования на Java 2. Кн. 2. Распределенные приложения / Х. Дейтел, П. Дейтел. – М.: Бином, 2003. – 464 с.

Поступила 25.03.08

Белорусский государственный университет,
Минск, пр. Независимости, 4
e-mail: dstrikelev@art-mission.com

D.A. Strykeleu

IMPLEMENTATION OF PARALLEL GENETIC ALGORITHMS WITH A COMPOSITE OBJECTIVE FUNCTION BASED ON JAVA RMI TECHNOLOGY AND JGAP LIBRARY

A library for Java genetic algorithms programming is considered and the integrated computational paralleling tools in heterogeneous network are investigated. The alternative program structure for the development of parallel genetic algorithms based on Java RMI technology is suggested and its efficiency for solving optimization tasks with composite objective functions is shown.