ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Факултет „Компютърни Системи и Технологии“

Катедра „Компютърни системи“

маг. инж. Аднан Бахри Реджеб

Методи за оптимизация на процеси и задачи при еднопроцесорни компютърни системи

АВТОРЕФЕРАТ

на дисертационен труд за присъждане
на образователна и научна степен «**ДОКТОР**»

Професионално направление:
5.3. Комуникационна и компютърна техника

Научна специалност:
Автоматизирани системи за обработка
на информация и управление

Научни ръководители*:*
проф. д-р инж. Огнян Наков Наков
доц. д-р инж. Румен Иванов Трифонов

София, 2017

Обща характеристика на дисертационния труд

Актуалност на проблема

Дисертационния труд разглежда проблем, чиято актуалност е безспорна поради факта, че касае паралелната обработка на информация, което е в основата на съвременните софтуерни системи, многозадачното управление, многопроцесната дейност на всяка една софтуерна система в съвремието, както и всяка една операционна система. Анализира методите, които са изполвани през годините, както и последните технологии използвани в развойните среди и компютърните системи. Систематизира подходите и предлага нови методи надграждащи съществуващите към момента, чрез съчетаване на познати и на не толкова популярни подходи, водещи до търсения ефект на дисертационния труд.

Цел и задачи на дисертационния труд

Целта на дисертацията е **разработване на методи за оптимизация, ускоряващи изпълнението на множество процеси и задачи в еднопроцесорни компютърни системи, чрез използване и надграждане на съвременни технологии за паралелна обработка на данни.**

За постигане на поставената цел се предвижда да бъдат изпълнени следните задачи:

1. Подробен анализ на класически алгоритми използвани при паралелните процеси;
2. Имплементация на анализираните алгоритми с цел съпоставка и сравнение с новите предложени в дисертационния труд;
3. Анализ на подходи за обхождане на структури (опашки съдържащи задачи);
4. Анализ на съвременни технологии използвани в развойни среди за паралелна обработка (Threads, ThreadPool);
5. Анализ на свойствата и капацитета на ThreadPool в различни развойни среди и избор на развойна среда за целите на експеримента;
6. Проектиране на метод за надграждане на ThreadPool, чрез предварителна промяна на реда на постъпващите в него задачи за изпълнение.
7. Имплементация на библиотека, чрез която се реализира надграждащият метод, както и класическите алгоритми за паралелна обработка;
8. Тест и сравнение на класическите подходи и новия надграждащ метод с различен брой задачи с различен приоритет и време за изпълнение;
9. Подробен сравнителен анализ и обобщаване на постигнатия ефект.

Приложимост и полезност

Дисертационния труд включва не само подробни анализи и предложени методи, а също така и софтуерна имплементация (библиотека за .NET среда) , която може да се използва при разработването на софтуерни системи изискващи обработване на множество задачи едновременно (паралелно). Методите са анализирани и имплементирани с цел анализ доказващ тяхната полезност и ефективност. За целите на дисертационния труд се използва конкретно развойна . NET технология, но са проектирани така, че биха могли да се имплементират лесно в други съвременни развойни среди и технологии.

Апробация

Резултатите от дисертационния труд са докладвани изцяло в катедра „Информационни Технологии“ на ФКСТ към ТУ-София. Основния метод е описан в статия на научно списание към ФКСТ „Computer and communications engineering“ със заглавие : „Метод за надграждане на ThreadPool в .NET среда при управление на паралелни процеси“. Останалите научни експерименти и анализирани методи са докладвани на научна сесия „Дни на науката 2016” на СУБ-Пловдив.

Обем и структура

Дисертационният труд е в обем от 168 страници и съдържа увод- въведение, основен текст структуриран в четири глави, заключение, списък с 5 публикации и използвана литература. Използвани са 28 фигури и схеми. Цитираните източници са 111.

Основни Глави

Увод

Глава 1 – Цели и задачи на дисертационния труд;

Глава 2– Обзор на съществуващите изследвания, методи и подходите за паралелна обработка ;

Глава 3 – Проектиране и имплементация на нови надграждащи методи, анализ и сравнения;

Глава 4 – Научни и научно приложни приноси;

Заключение

Списък на публикации

Литература

Глава I : Цели и задачи на дисертационния труд

Главата разглежда целите и задачите на дисертационния труд, които вече бяха споменати по-горе;

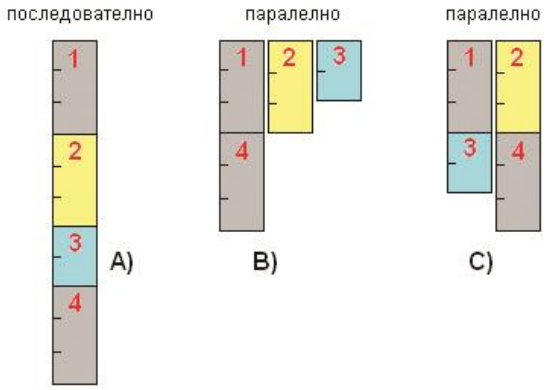
Глава II: Обзор на съществуващите изследвания, методи и подходите за паралелна обработка

В Глава II са анализирани основни класически методи за паралелна обработка на информация. Освен подробния анализ е реализирана и имплементацията им на база описаните алгоритми, което е принос на дисертационния труд.

N-мерен паралелизъм

Характеризира се със зависимост между данните в последователността за изчисление, потенциално определяща едновременните процеси. При него възможностите за паралелна обработка на данните в приложенията са ограничени, заради необходимостта от последователно изпълнение на някои данни. Нека разгледаме:

Представяне на последователно и паралелно изпълнение при възможност за N – мерен паралелизъм.



Фиг. 1

Конвейерен паралелизъм

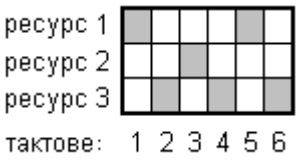
Следващият вид паралелизъм който ще разгледаме, според класификацията си в приложенията е конвейерен. Конвейерният паралелизъм е специфичен. В своята същност, той използва обработка на фрагменти от код, които имат нужда от достъп до различни ресурси в определена последователност. Конвейерният паралелизъм се грижи ресурсите да се използват пълноценно, като всеки ресурс който се освобождава от един процес или фрагмент от код, да се предоставя на друг процес или фрагмент. Конвейерният паралелизъм се разделя на две под групи: линеен и нелинеен.

В линейният конвейерен паралелизъм всички ресурси се използват само в определен ред и най често се обработват само задачи със сходна последователност на използваните ресурси. При конвейерната паралелна обработка на процеси е възможно, те да не са еднакви. Подобни случай са характерни за нелинейният паралелизъм, който следва да разгледаме.

Освен показаните до момента линейни конвейери, в които изпълняваните приложения и задачи имат проста структура, е възможно задачите да имат сложна структура, в която последователност на операциите пречи на работата на конвейера. Също така е възможно структурата на задачите да бъде различна, с което работата на конвейера се затруднява допълнително. Сложната структура на задачите за изпълнение в конвейерна обработка се изразява в използването на един ресурс повече от веднъж, с което се внася допълнителна зависимост в конвейерната подредба на задачите за изпълнение.

Множество задачи с еднаква структура се изпълняват в конвейерна последователност. Задачите използват три ресурса на КА върху, която се изпълняват, като някои от ресурсите се използват повече от веднъж, за изпълнението на всяка от задачите.

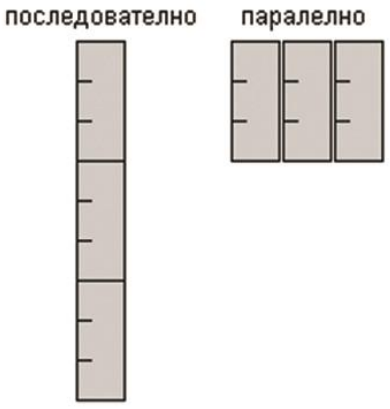
В Таблицата 2 е показана една задача и последователността, в която тя използва различните ресурси на КА. Продължителността на задачата е 6 работни такта или ЕМВ, в които протича нейното изпълнение.



Фиг. 2

Векторен паралелизъм

Векторният паралелизъм се използва, когато имаме голям брой еднотипни данни, като съдържанието на един масив и върху тях трябва да извършим една операция, като примерно да ги повдигнем на квадрат. Всяка от данните в масива може да се пресметне паралелно на останалите, в отделен процес. При наличие на достатъчен брой процесори всички данни могат да се пресметнат едновременно, както е показано на Фиг 3



Фиг. 3

Характерно за трите вида паралелизъм: N-мерен, конвейерен и векторен е, че те се използват в приложения с различна грануларност на паралелната обработка.

N-мерният паралелизъм се прилага в приложения със средна и груба грануларност. Приложения с фина гранулярност, са съставени от малки сегменти от код и за тях е характерно, да нямат зависимост между отделните сегменти.

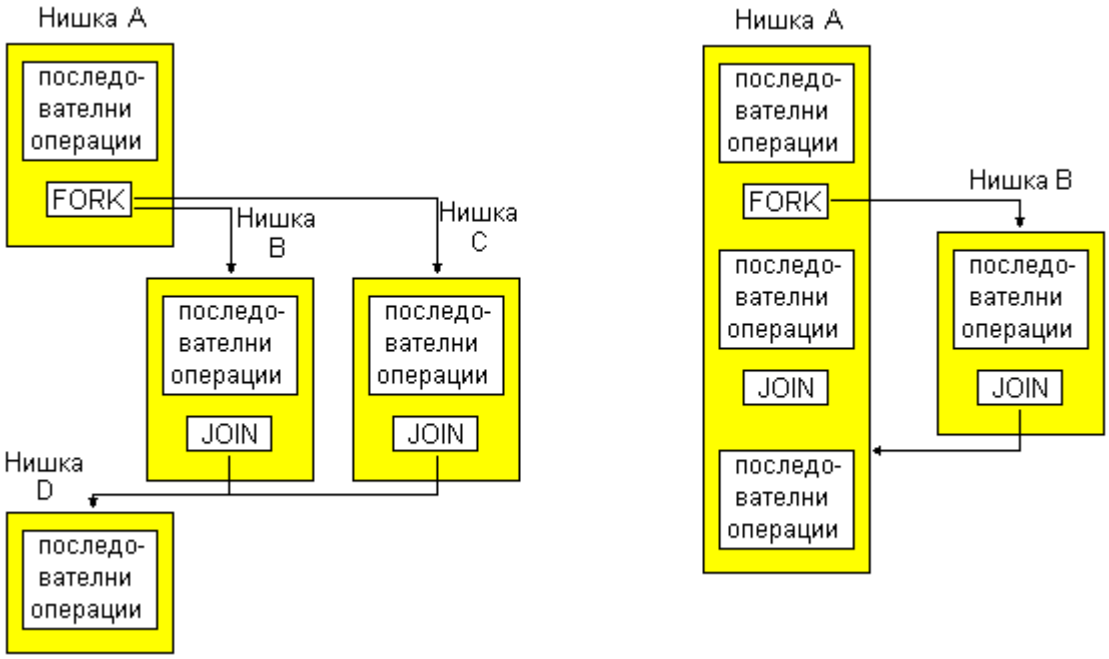
Реализация на паралелизма в абстрактните модели на КА

След като се запознахме с различните видове паралелизъм, приложението им в програмите и специфичните им характеристики, е необходимо да разгледаме как се реализира паралелизма в КА. За целта ще разгледаме класификацията на абстрактните модели на КА.

КА могат да работят с паралелно или последователно изпълнение на задачите, които им се поставят. Последователното изпълнение в КА, може да се осъществи използвайки две основни техники: програмен брояч и микро програмно управление.

КА с последователното изпълнение не са цел на нашето изследване, заради това ще се ограничим само с кратка информация за тях. По-важни са КА предназначени за изпълнение на паралелни задачи, заради което ще обърнем повече внимание, на техните абстрактни модели. Характерно паралелното управление на фрагменти от една задача е необходимостта от моменти в които се дефинират началото и края на паралелното изпълнение. Паралелното изпълнение на фрагменти код от един процес става посредством различни нишки. Началото на паралелно изпълнение е подобно на разклонение (fork), след което програмата започва да се изпълнява в различни нишки. Когато работата на два или повече процеса, изгуби възможност за паралелна обработка те преминават към последователна, като се присъединяват в една нишка. Понякога паралелните процеси могат да приключат с изпълнението на работата си и да прекъснат нишката която са използвали.

От използването на една на една нишка трябва да преминем в използването на две. Нишката, която имаме в началото, може да създаде две нови нишки и да прекрати своето съществуване, (*Схема 6*). Възможно е и нишката, която имаме да изгради само още една нишка и двете да работят паралелно каквото е изискването (С*хема 7*).



Фиг. 4

Същност на "Thread pool"

Разглежда се основния обект, който се надгражда в дисертационния труд и за който е предназначена имплементацията на библиотеката в . NET среда.

Thread Pool е група от нишки, които могат да се създават от ОС и да се използват от различни приложения, или да се създават от конкретно приложение и да се използват от различни негови процеси. Основно правило за нишките в тази група е, че те се създават и остават притежание този който ги е създал, но се предоставят за работа на други задачи. След приключване на работа на задачите, които използват нишките, нишките не се разрушават, а се връщат на собственикът им. По този начин се спестява време за създаването и за разрушаването или освобождаването на нишките, с което се ускорява работата на приложенията. Освен нишките, които се предоставят за работа с други задачи, имаме и една основна нишка, която се грижи за правилната работа на групата от нишки. Тази основна нишка създава останалите нишки от групата и ги управлява. Когато една задача поиска управлението над нишка от групата, основната нишка предоставя една от останалите нишки, ако това е възможно. По този начин основната нишка остава свободна за обработка на други заявки и допълнителни асинхронни действия, по поддръжката на групата от нишки.

Групата от нишки работи подобно на фонов процес. Нишките, които не се ползват в определен момент, са създадени и готови за работа, но а тях не се предоставят интервали от време за работа с ресурсите на КА. При постъпването на нова задача, на нея се предоставя самостоятелна нишка, като по този начин задачата може да се изпълнява паралелно и асинхронно на останалите задачи. Използването на подобни методи е характерно за сървърни приложения, към които се отправят множества задачи, които трябва да се изпълняват паралелно и асинхронно.

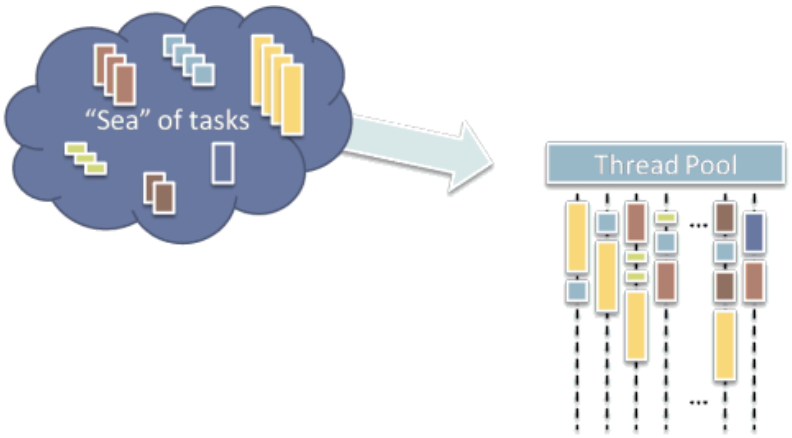
Невинаги постъпилите задачи, могат да получат достъп до нишка. Групите от нишки или Thread Pool обикновено имат ограничение в максималният брой на създадените нишки.

Ако максималният брой нишки за създаване е достигнат и всички нишки се използват в момента на постъпване на нова задача, то новата задача се подрежда в опашка, от която изчаква реда си за получаване нишка с която да работи. Задачите в опашката получават нишка с която да работят в реда на пристигането си, като това става, когато някоя задача приключи работата си и освободи нишката която се използвала. Докато задачите са в опашката, те не могат да бъдат обслужвани, като по този начин се забавя тяхното изпълнение.

Въпреки ограничението на максималният брой на нишките в групата, това не е голям проблем. Максималният брой може да бъде регулиран. И въпреки съществуването на ограничение, то не се достига често. По често явление за групата от нишки е да се използва непълният капацитет на групата, като по този начин нишките, които са създадени в повече ангажират част от ресурсите на КА и пречат на работата на ОС. За да се справим с този проблем, групата от нишки се прави динамична, като броят на нишките се променя според броят на постъпващите задачи.

Важен момент, който трябва да отбележим е, че въпреки промяната на броя на нишките, те не се създават и разрушават при началото и края на работа на една задача с тях. Ако се случваше това, нямаше да има полза от използването на група от нишки. Този вариант дори би имал негативен ефект, заради използването на допълнителна нишка за управление на групата, ограничение на групата и на опашка в която задачите са принудени да изчакат свободна нишка. За да се избегнат всички тези негативи, броят на нишките от групата варира спрямо броят на постъпващите и обработваните задачи. Като целта е едновременно да се използват нишки, които не се изграждат и унищожават индивидуално за всяка задача и същевременно броят на нишките да не е много по-голям от броя на изпълняваните задачи. За да се постигне равновесие между двете условия се използват различни методи, като тук се намесва и нишката която отговаря за управлението на групата.

Едната възможност, която се използва за оптимизиране на заетостта на групата от нишки, се прилага в случаите, в които е необходимо да не се изчаква създаването на нишка при началото на работа на нова задача. За да започне по-бързо обработката на новопостъпилата задача, се използва една от вече създадените нишки. След което основната нишка създава асинхронно нова нишка. По този начин новопостъпилата задача може директно да пристъпи към изпълнението си. За да се постигне този ефект броят на създадените нишки е по-голям отколкото броят на задачите, които се изпълняват в момента. Това гарантира един интервал от няколко нишки който могат да се предоставят веднага при постъпване на нови задачи. Една от възможностите, с които може да се направи тази оптимизация е използването на минимум от свободни нишки. Въпросният минимум се подържа от основната нишка, която при необходимост създава нови нишки.

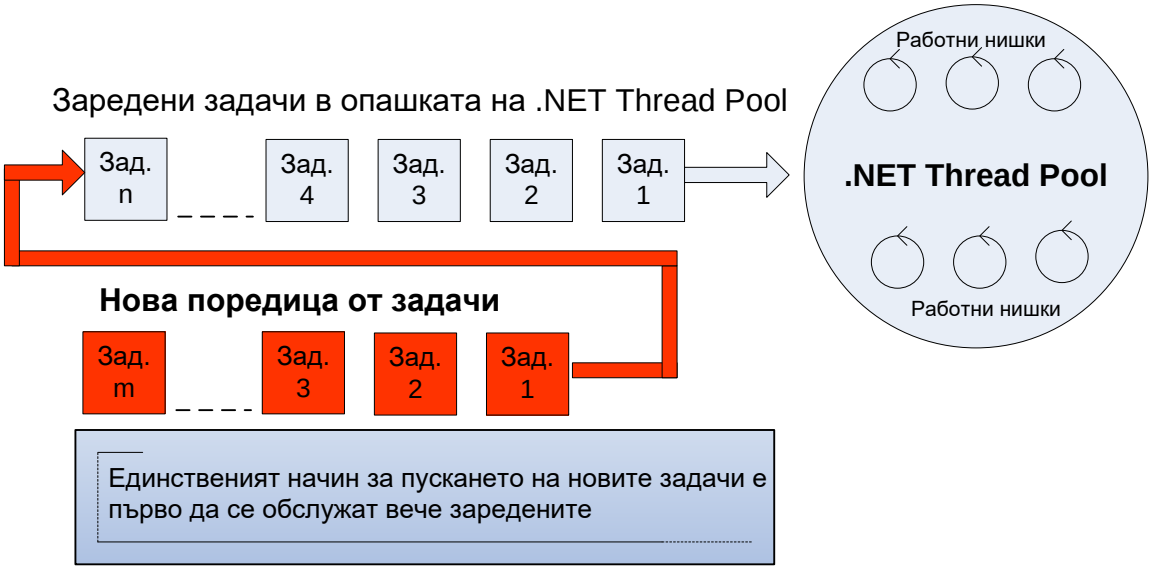


Фиг.5

Глава III : Проектиране и имплементация на нови надграждащи методи, анализ и сравнения

Глава III е основна за дисертационни труд и в нея е описан основния предложен метод за надграждане на TreadPool, чрез пренареждане на входните задачи в нови структури и подредени по различни критерии. Предложени са няколко метода за обхождане на новите структури. Следва имплементация на библиотека, която реализара метода за .NET среда, както и имплементира и класическите методи с цел сравнение, анализ. За целта се извършват редица тестове с различен брой и тип задачи.

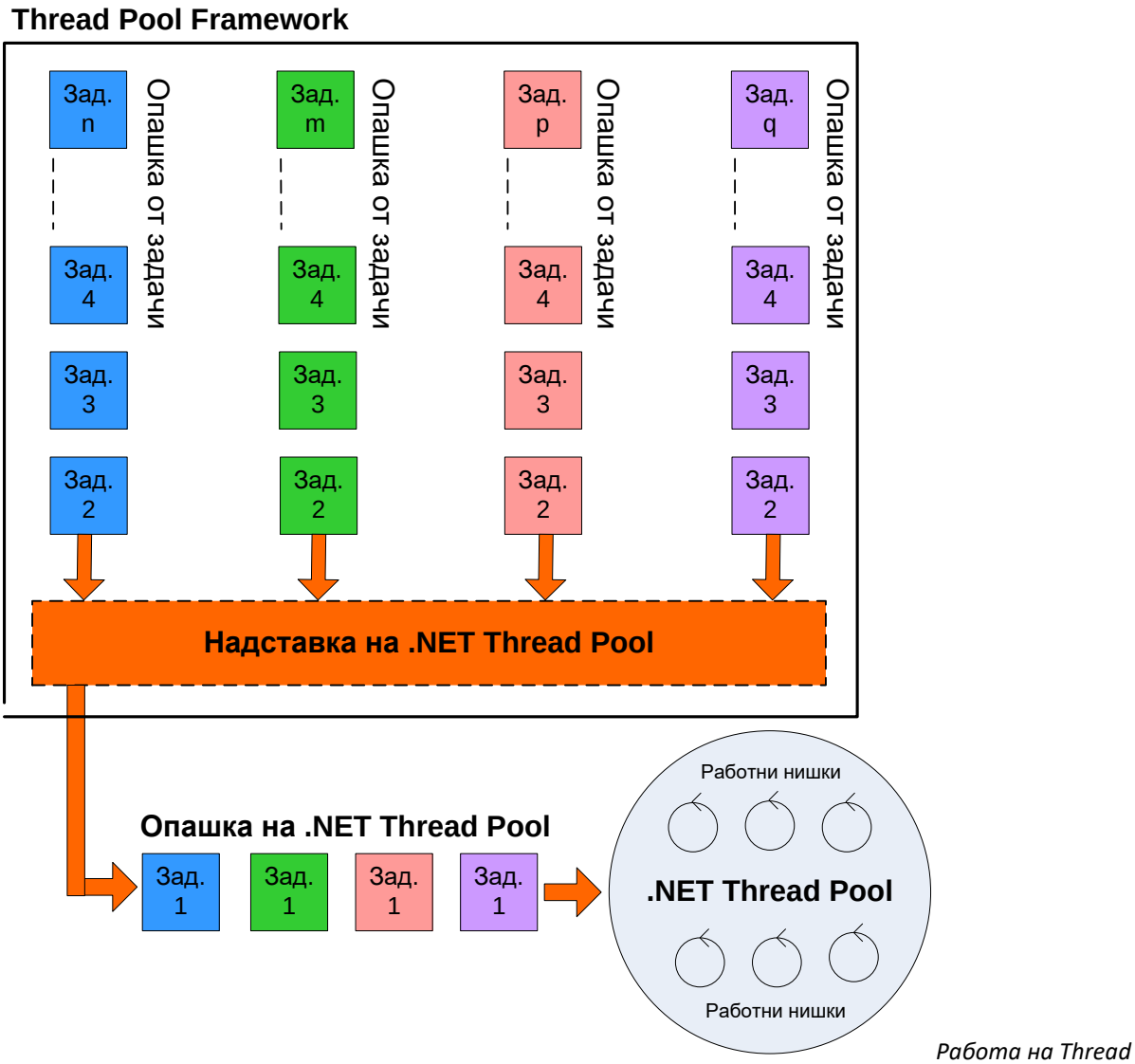
Добър пример за проблема е приложение, което има нужда от използване на .NET Thread Pool, за да обработи голям брой задания асинхронно. Работата на Pool-а ще бъде подадена по стандартния и единствен начин – опашката със задачи ще се зареди. Но ако в даден, малко по-късен момент, пристигне друга поредица от задачи, тя ще трябва да изчакат пускането на вече заредените. Новите нямат никакъв шанс да се включат по-рано от последната, заредена вече задача.



Фиг. 6

Първата и основна цел е предлагане на начин за управление на реда на изпълнение на задачите. Идеята е това да стане с възможност за работа с повече от една опашки от задачи. В горния пример, ако опашките от задачи са две и заданията се пускат последователно, от всяка опашка по едно, така ще се избегне нежеланото чакане на втората поредица. Алгоритъмът за взимане на задачи от опашките ще бъде кръгова дисциплина на обслужване (Round Robin).

Първата и основна цел е предлагане на начин за управление на реда на изпълнение на задачите. Идеята е това да стане с възможност за работа с повече от една опашки от задачи. В горния пример, ако опашките от задачи са две и заданията се пускат последователно, от всяка опашка по едно, така ще се избегне нежеланото чакане на втората поредица. Алгоритъмът за взимане на задачи от опашките ще бъде кръгова дисциплина на обслужване (Round Robin).



Работа на Thread Pool Framework с повече от една опашка

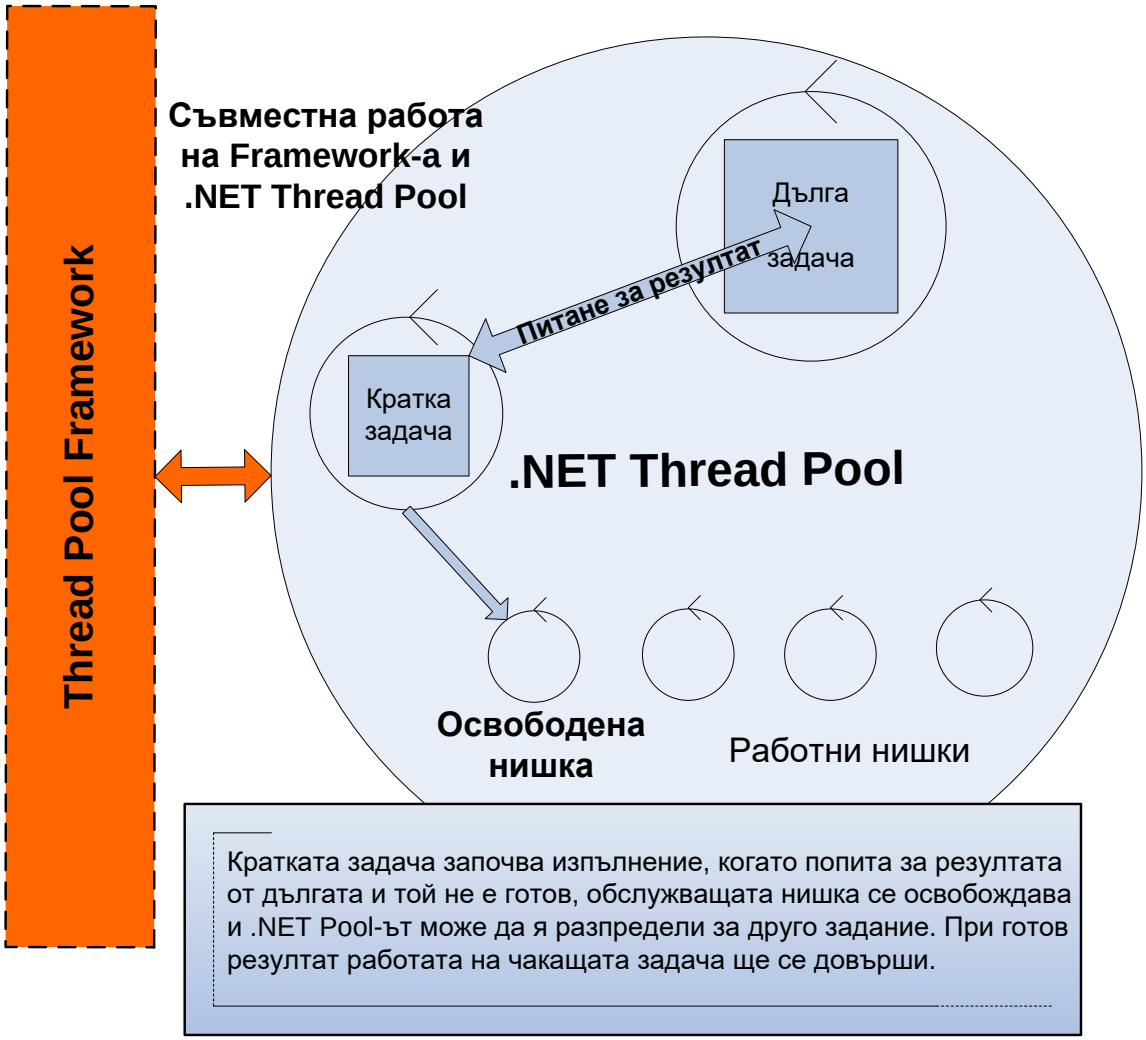
Фиг. 7

Round Robin (RR): в контекста на работа се има предвид взимане на една задача от всяка опашка и пускането ѝ за изпълнение от .NET Thread Pool. Опашките се обхождат последователно, една след друга, докато има задачи в тях.

Пример: В по-горното визуално представяне на работа има четири опашки от задачи, от всяка се взима по една задача – първо от синята, после от зелената, розовата и лилавата. В този ред задачите се подават на .NET Pool-а. Тези итерации по взимане на задания се извършват докато се изпразнят всички опашки.

Следващият важен оптимизационен метод е осигуряване работата на взаимно свързани задачи – една задача чака резултат от друга, за да продължи изпълнението си. Този сценарий също не може да бъде реализиран с функциите предлагани от Pool-а на .NET. Това е удобно когато дадена задача се изпълнява продължително, а друга, значително по-кратка за изпълнение, чака резултат от нея. В този случай нишката, ангажирана за обработка на краката задача, ще трябва да стои и чака резултата, за да довърши работата си. Идеята тук е да се освободи работната нишка на кратката задача, така че да може да бъде разпределена от Pool-а за друго изпълнение. Щом резултатът

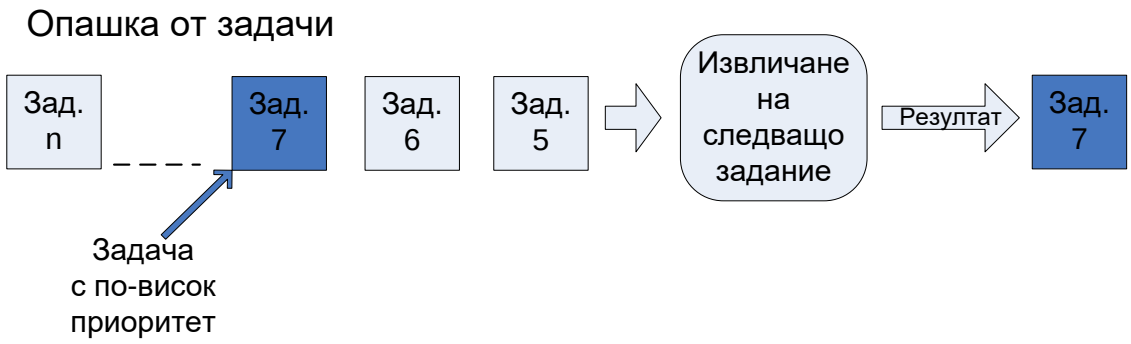
от дългата задача е готов, довършителната работа на късата задача ще бъде изпълнена. По този начин се избягва излишното бездействие от работните нишки при чакане на резултат.



Начин на работа при взаимно свързани задачи

Фиг. 8

Предвидено е и задаване на приоритет на задачите, засягащ извличането им от работните опашки на новия framework. Всяко задание може да бъде подавано с приоритет. Задачите с най-голям приоритет ще бъдат взимани първи от опашката. Приоритетността също липсва в .NET Thread Pool.



Приоритетна работа на опашките

Фиг. 9

Framework-ът ще осигурява и прихващане на изключеня (exceptions). Това е в случай, че при изпълнение на задачите, възникне изключение, което не е прихванато в потребителския код. Предвидено е събитие, чрез което ползвателите (програмистите) могат да бъдат информирани за изключението и да предприемат някакви действия, например записване на грешката (log, logging).

Относно работата на опашките ще има два режима. Първият е нормален – задачите се взимат последователно, когато дойде ред за пускане на задача от съответната опашка. Това зависи от алгоритъма с кръгово разпределение (Round Robin). Другият режим предоставя различно поведение – в опашката могат да се подават задачи, но те не се обработват, кръговото обхождане на опашките пропуска извличането на задачи от тях. По-късно програмистът има възможност да включи опашката за обработка. По този начин могат да се събират задания, но да се обработват в желан от потребителя момент.

<u>Обобщените цели и задачи на разработката са имплементиране на framework, включващ:</u>

- възможност за разпределение на заданията в различни опашки, които се обработват с алгоритъма Round Robin
- осигуряване на работата на взаимно свързани задачи (задача чака резултат от друга) и избягване на излишното чакане от страна на работните им нишки
- възможност за задаване на приоритет на задачите
- прихващане на изключения при изпълнение на задачите и нотификация за изключение чрез събитие
- два работни режима за опашките – задачите се пускат за обработка веднага или се събират и стартират по-късно

Имплементират се посочените известни към момента методи в среда на високо ниво , а именно .NET ,

Проектиране

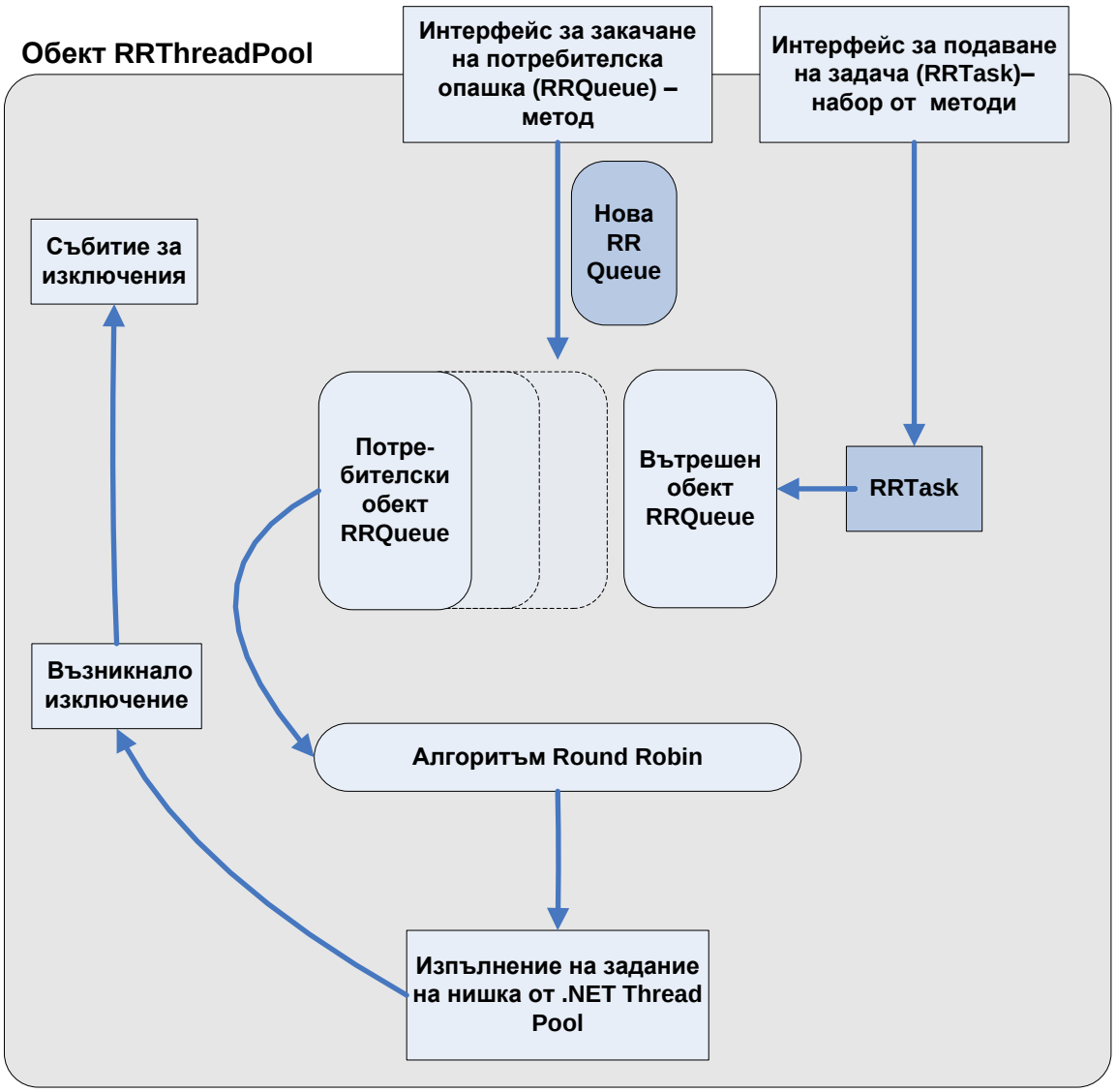
За да се реализира успешно какъвто и да е framework е необходимо добре да се обмислят основните функции, проблемите които ще решава. Важно е да се намерят качествени и оптимални решения на дефинираните задачи. Подходът на работа е малко по-различен в сравнение с класическите приложения. Под класически се разбират ориентираните към масовия потребител – счетоводни програми, програми за управление на бизнеса, web сайтове и др. Всички те работят с потребителски интерфейс и строго дефинирани изисквания от клиента. Съответно етапите при разработка са съвсем други. Тези програми основно работят с данни (системи за релационни бази от данни, XML и др.), насочени са към специфичните тема и функционалност, фокусират се върху добър контрол на входните и изходните данни, добра модулна структура, изискват специален подход по разпространение и вграждане на системата.

При създаването на конкретен framework, акцентите при проектиране и разработка падат върху качеството, осигуряването на стабилна работа, прихващането на всички възможни случаи и сценарий. Потребителите на приложението ще бъдат програмисти. Трябва да се обмисли подходящ, лесен и удобен интерфейс – именоване на класове, методи, функции. Целта е с минимални усилия да бъдат предоставени възможностите на библиотеката.

При стартиране на работата беше определена конвенцията за именоване, която се спазва в целия програмен код. Използвана е така наречената CamelCase конвенция, известна като medial capitals (междинни главни букви). Тя налага главни букви за всеки един елемент от даденото име. Например ако името на даден клас е от две различни думи, то те се сливат, като всяка започва с главна буква – ThreadPool (thread pool), метод QueueTask (queue task), метод AttachQueue (attach queue). Имената на всички основни обекти започват с префикс RR (от Round Robin). За пространството от имена е избрано името RRTasking.

При проектирането се определя архитектурата на основните обекти, нужни за предоставяне на дефинираната функционалност. В случая, най-важните обекти са три – RRThreadPool, RRQueue, RRTask. Важно е преди имплементация на framework-а да бъдат определени основните им функции и роля, да се измисли и предложи вариант за постигане на целите.

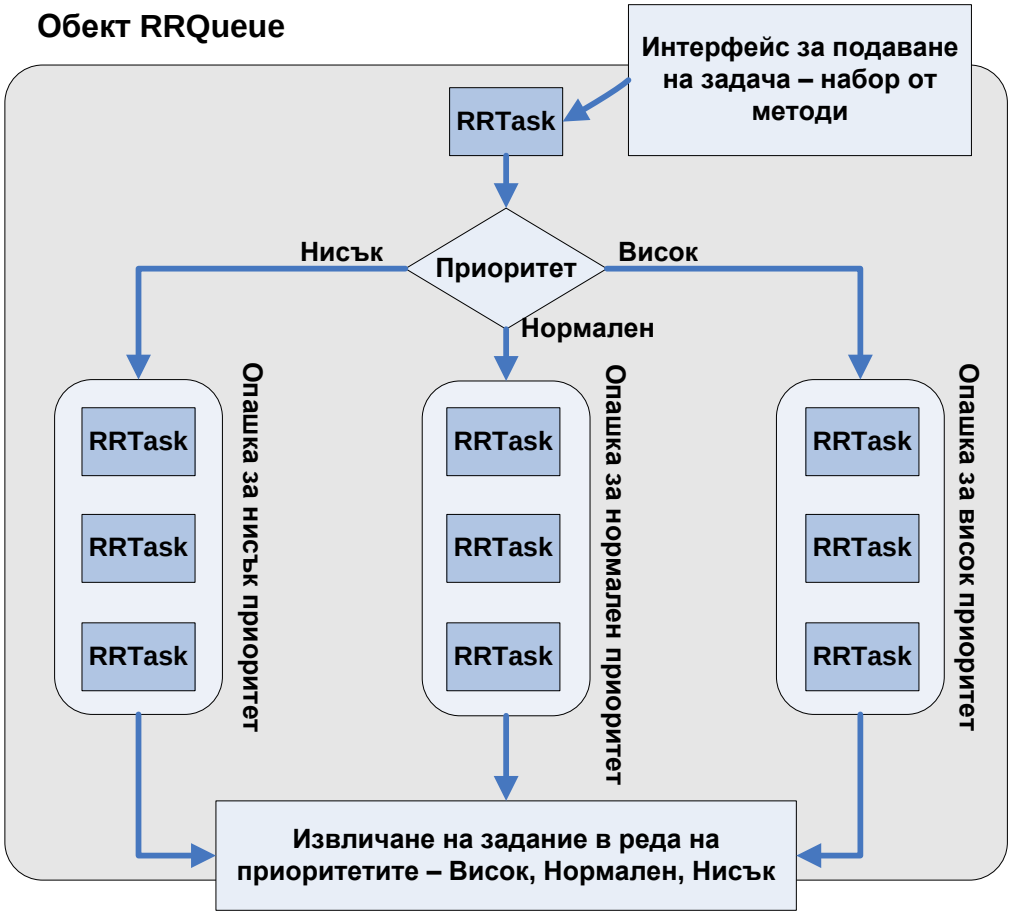
RRThreadPool е основният обект. Той представлява абстракция на контейнер с нишки. Основната му функция трябва да бъде разпределяне на задачи от опашките по алгоритъма round robin и пускането им за изпълнение от .NET Thread Pool. Всъщност, той вътрешно поддържа списък от опашки, но е предвиден интерфейс за подаване на задачи директно на RRThreadPool. Това означава, че трябва да има поне една вътрешна опашка, използвана като служебна и скрита за потребителя. В този клас ще бъде дефинирано и потребителското събитие, което ще се използва за нотификации при изключения (exceptions). Алгоритъмът за извличане на задачите (round robin) трябва да се съобразява с двата режима на работа на опашките. По-долу е дадена диаграма, показваща основните задължения на този обект.



Основни задължения на обекта RRThreadPool

Фиг. 10

RRQueue е другият важен обект. Той трябва да организира задачите, подредени в структура опашка. Обектът се използва от RRThreadPool. Заданията се извличат по приоритет на ниво опашка, затова тук трябва да бъде реализирана приоритетността. Избрано е задачите да могат да имат три възможни приоритета – нисък (low), нормален (normal) и висок (high). За да стане това, обектът RRQueue ще капсулира три опашки от framework-а за колекции на .NET. Всяка една ще отговаря за дадения приоритет (нисък, нормален или висок). Когато се подаде задание на опашката RRQueue, в зависимост от приоритета, то ще се поставя в съответната вътрешна опашка. При поискване на задача ще се преглеждат трите опашки в ред от най-висок към най-нисък приоритет. Така задачите с най-висок приоритет ще бъдат давани първи, задачите с нормален – втори и задачите с нисък – последни.

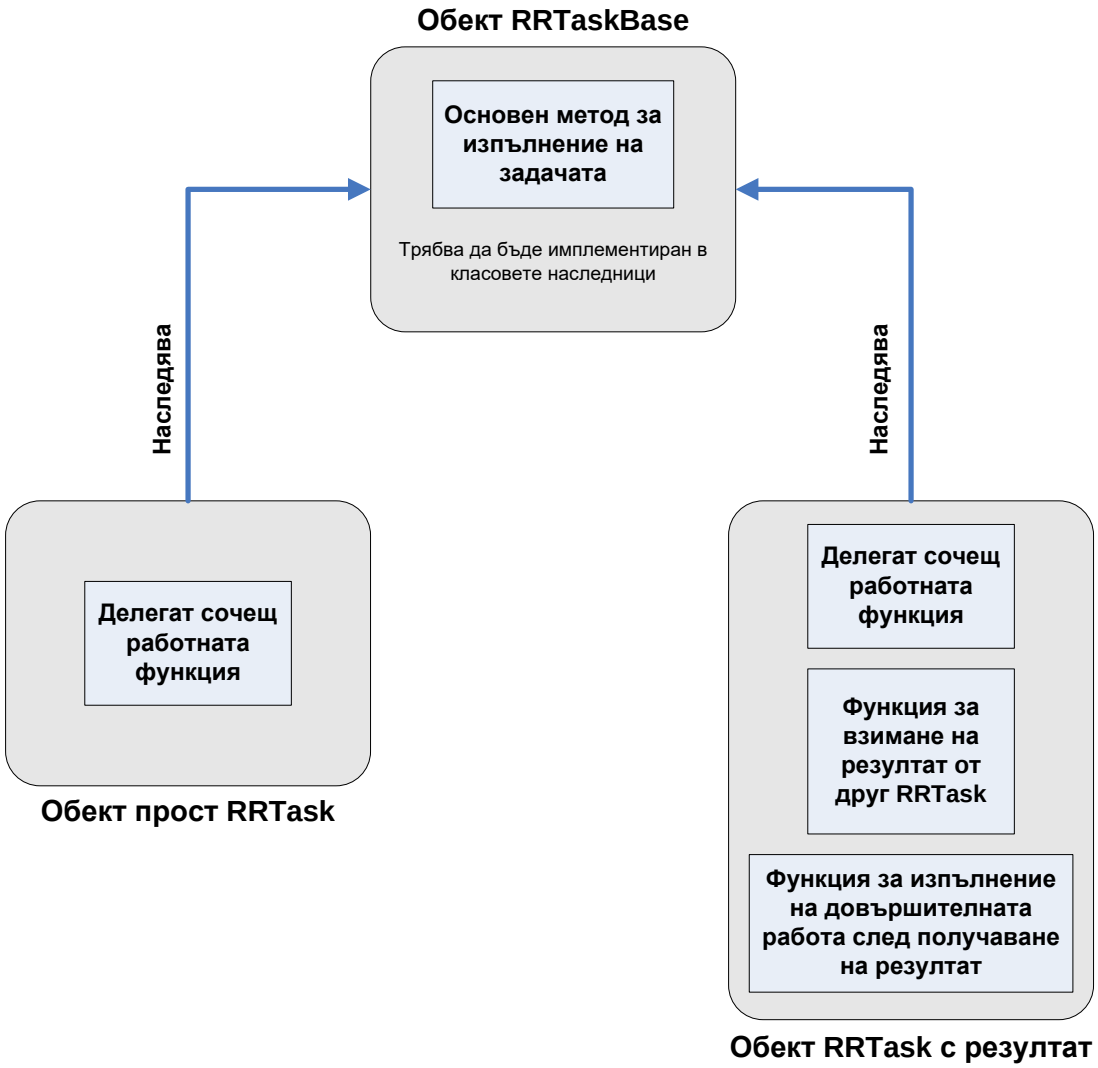


Начин на работа на обекта RRQueue

Фиг. 11

RRTask е обект, представяш потребителско задание. От дефинираните изисквания става ясно, че този обект е малко по-специален. Освен проста задача, той трябва да представя и взаимно свързаните задачи, при които някои могат да чакат резултат от друга, нужен за довършване на работата им. За да може да се постигнат тези изисквания е предвиден базов клас – RRTaskBase, който ще се наследява от различните RRTask класове. Цялостният framework ще работи с RRTaskBase, което ще дава възможност за добавяне на различни по тип задачи (класове), ако потребителят има нужда от специфично поведение на работа. В текущата имплементация ще има задачи, представяни от два класа, които наследяват RRTaskBase. Първият представлява проста задача, тя просто се подава и изпълнява от Pool-а. Вторият ще бъде задача, която има допълнителни функции – за искане на резултат и логики осигуряващи изпълнението на довършителната работа след получаването на резултата.

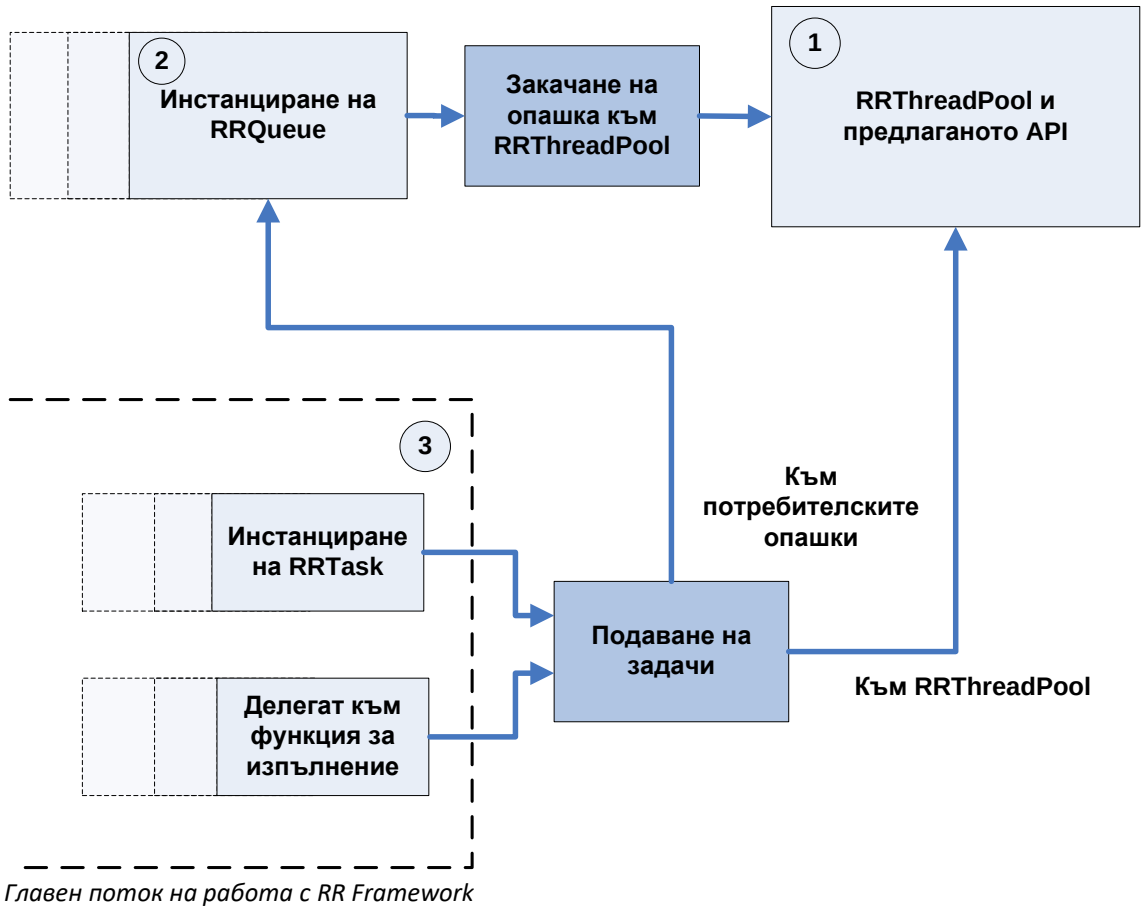
Тъй като базовият клас ще може да бъде наследяван от класове в потребителския код, необходимо е да се обмисли внимателно дефиницията му. Той трябва да бъде прост и да задължава програмистите да имплементират необходимите му за правилна работа функции. Такава функция (метод) е този за конкретно изпълнение на задачата, най-често това е извикването на делегата, указващ функцията, в която е програмният код за изпълнение от Thread Pool-а.



Йерархия и основни функция на обектите RRTask

Фиг. 12

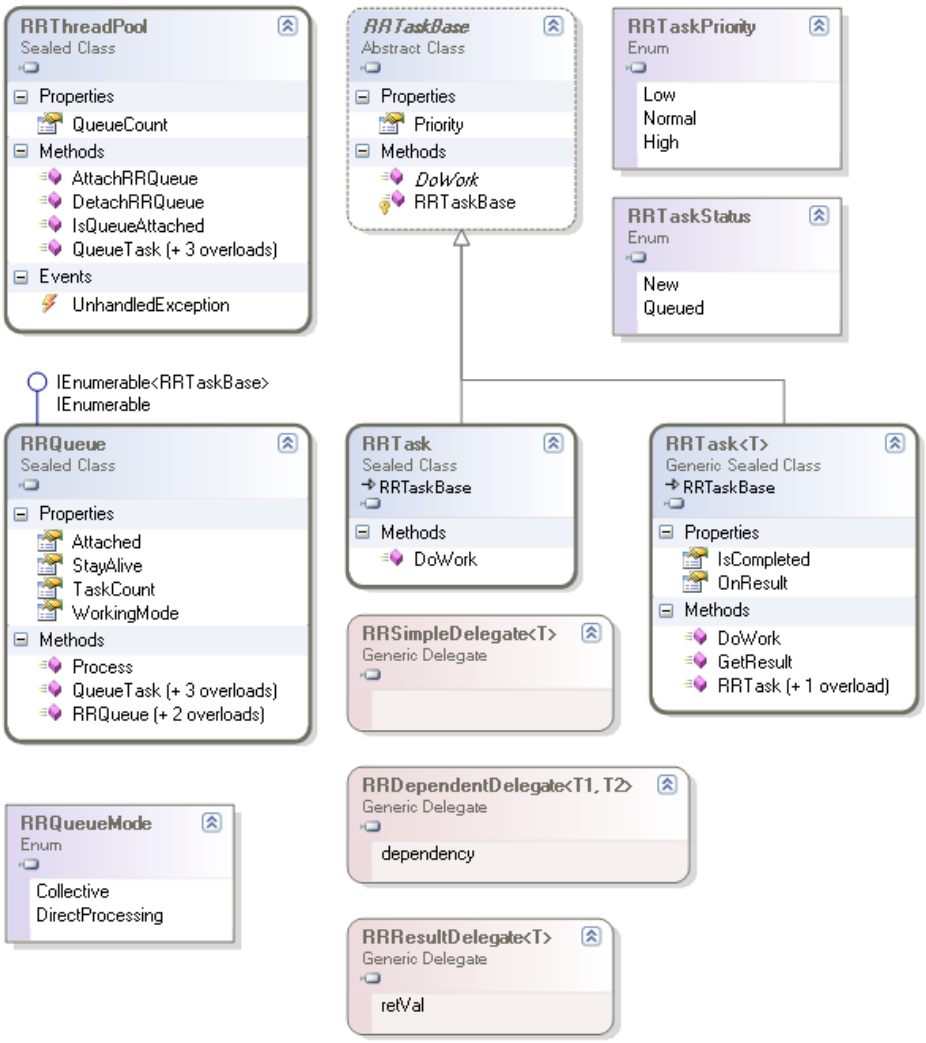
Най-важните функции и предназначения на главните обекти са изяснени. Друг важен етап при проектирането е да се щрихова начинът на работа с RR Framework-а. Има се предвид от потребителска гледна точка. Трябва да е ясно какви инстанции на обектите ще са нужни, какъв ще бъде потокът на работа (work flow) Най-често използваният начин на работа ще бъде с методите на RRThreadPool и RRQueue. RRQueue обектите трябва да се закачат за Pool обекта, това означава, че те ще бъдат управлявани от него при пускането на задания (алгоритъм Round Robin). Задачите ще могат да бъдат подавани както на RRQueue обектите (потребителските опашки), така и на RRThreadPool, като той ще ги обработи чрез вътрешната си опашка. Закачането на опашка и подаването на задача ще става с методи на обектите RRThreadPool и RRQueue.



Фиг . 13

След като са уточнени всички изисквания, цели и задачи, изяснена е основната структура, избрани са начини за постигане на целите, може да се пристъпи към програмната реализация.

Имплементация на класически методи, се интегрират в библиотеката.
Основни класове на създадената библиотека :



Фиг. 14

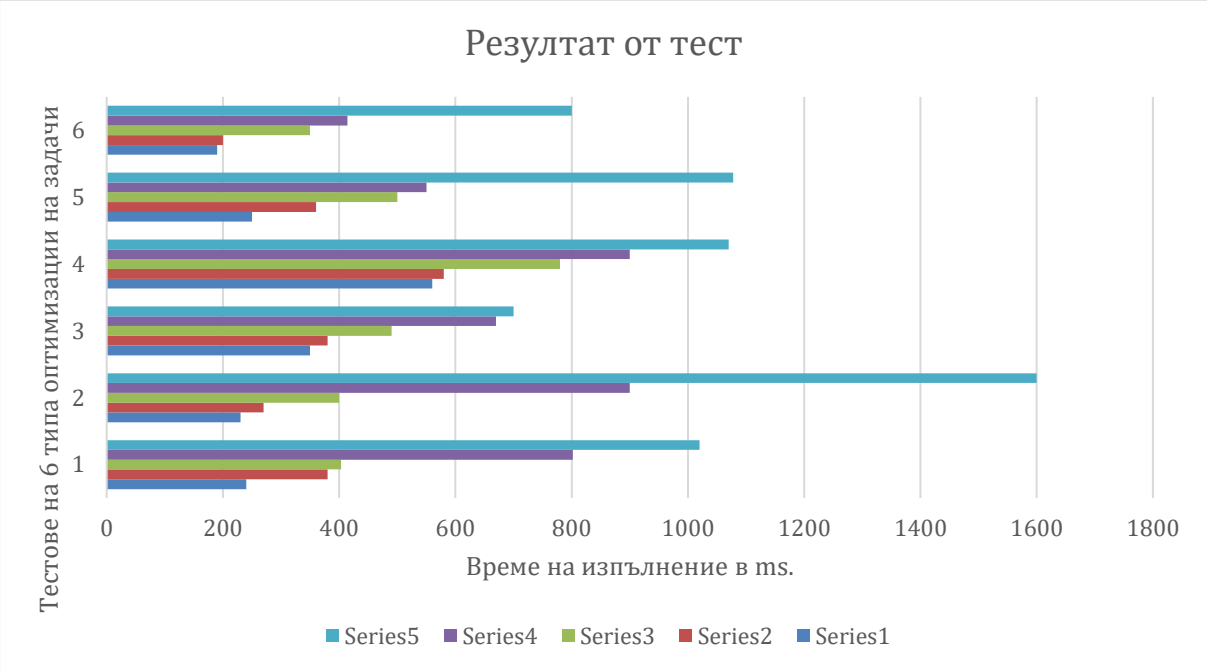
Следва Експерименталната част :
ТЕСТ и АНАЛИЗ

Извършени са няколко теста с различен брой задачи посредсвтом библиотеката и ThreadPool в . NET среда.

Брой задачи	N-мерен паралелизъм	Конвейерен паралелизъм	Векторен паралелизъм	Псевдо паралелизъм в еднопроцесорни КА	Кръгова обработка на задачите	Надграждащ метод	
2	240	230	350	560	250	190	ms
3	380	270	380	580	360	200	ms
15	403	400	490	780	500	350	ms
30	802	900	670	900	550	414	ms
50	1020	1600	700	1070	1078	800	ms

Таблица 1

Експеримента се извършва в следната последователност, стартира се симулационна среда, която генерира задачи с различен приоритет, който се задава на случаен принцип, както и задачи с циклични структури и логически операции с различна сложност. Генерираните задачи се обработват от имплементираната библиотека, подреждат се в 3 или повече опашки по различен критерии. Следва обхождане на опашките в кръгова дисциплина и преподреждането им при влизане в ThreadPool. Създадената библиотека записва времето за изпълнение в ms за всеки следващ тест. Тестовете се извършват на база различно преподреждане на задачи по критерий време и приоритет, различен ред на опашките, низходящо, възходящо и смесено подреждане на опашките. Тестове се извършват и чрез класическите алгоритми. Измервания се извършват с различен брой задача с цел да се определи скалируемостта на предложения метод.



Фиг. 15

Изводи

В заключение, може да се твърди, че прилагането на надграждаща библиотека ускорява дейноста на ThreadPool-а в .NET среда. Експериментът може да се проведе и чрез технологии като Java, като се използва Java ThreadPool-а. Предвид различната имплементация на двата Pool-а, то резултатите биха били различни, както и мониторинга в реално време на обработката на информация. Класическите методи за разпаралеляване, работят значително по-бавно от предложения метод надграждащ ThreadPool-а, както и употребата на нишки, които не са обединени в Pool.

Информация за експерименталната среда

Експерименталната среда указва силно влияние върху измерените резултати, затова е важно да се докладват системните ѝ характеристики и версиите на използваните приложения. Те биха

били необходими за репликация на експерименталните установки в друга среда и корелация на получените резултати:

- **Системните характеристики** на работната станция използвана за изпълнение на експериментите са: **процесор** Intel(R) Core(TM) i5-4200M CPU @ 2.50 GHz с 64-битова 22 nm архитектура, общо 4 ядра (две физически плюс HyperThreading) и кеш L1 / L2 / L3: 64 KB / 256 KB / 3072 KB; **памет**: Lenovo 16 GB DDR3L (2 x 8 GB PC3-12800 SODIMM модула) със скорост 1600 MHz; **твърд диск** Western Digital WD10JPVX 2.5 инча с капацитет 1 TB, интерфейс SATA, скорост 6 Gb/s и кеш 8 MB; **дънна платка** Intel NM-A161.
- Програмното осигуряване Експериментални резултати по фаза – Средата за разработване на библиотеката е .NET, задачите са циклични примери, както и други подобни кратки логически конструкции. Приоритета е зададен предварително, а времето на изпълнение се определя от степента на сложност на задачите.

Глава III . Приноси на дисертационния труд

Приносите на настоящата дисертация са групирани в две категории:

Научни приноси

- Подробен анализ и синтезиране на основни класически алгоритми за разпаралеляване на процеси и задачи;
- Анализ на алгоритми за обхождане на структури (контейнери на задачи) ;
- Предложен нов надграждащ .NET ThreadPool метод, на база извършените анализи;
- Сравнени на класическите и проектирания от дисертанта метод;

Научно-приложни приноси

- Имплементация на .NET библиотека предлагаща новия метод за надграждане;
- Имплементация на N-векторен, Конвейрен и Векторен алгоритъм за разпаралеяване на процеси;
- Изграждане на цялостна тестова среда за изпълнение на множество задачи;
- Подробен сравнителен анализ на извършените тестовете и доказване на твърдението за ускорение на обработка чрез преподреждане на първоначалния ред от задачи.

Заключение

Разработеният метод имплементиран в библиотека на . NET, както и останалите класически методи, могат да бъдат прилагани в редица програми използващи паралелни изчисления.
Авторът на дисертационния труд планира имплементация на подобна библиотека чрез използването на други среди (Java) и извършване на тестове на устройства с по-малки изчислителни капацитети и ресурси. Предвиждат се тестове и в WEB среда.

Списък с публикации по дисертацията

1. ***Самостоятелна статия*** : ***Аднан Реджеб*** , Метод за надграждане на ThreadPool в .NET среда при управление на паралелни процеси, **списание** Computer and communications engineering 1/2017 (приета за печат януари 2017);

2. Roumen Trifonov, Ognian Nakov, ***Adnan Redzheb*** ,CLOUD COMPUTING E-GOVERNMENT APPLICATIONS AND INFORMATION SECURITY , **конференция** Advanced Aspects of Theoretical Electrical Engineering Sofia '2016 15.09.16 – 16.09.16, Sofia, Bulgaria;

3. Ognian Nakov, Roumen Trifonov, ***Adnan Redzheb*** , MODELING AND MANAGEMENT OF SERVICES IN THE FRAMEWORK OF THE SERVICE ORIENTED ARCHITECTURE, **конференция** Advanced Aspects of Theoretical Electrical Engineering Sofia '2016 15.09.16 – 16.09.16, Sofia, Bulgaria;

4. ***Adnan Redzheb***, Ognian Nakov, A STUDY ON A CONVEYER PARALLELISM WITH THE USE OF FIXED WEIGHT COEFFICIENT, **конференция** „ Дни на науката“ 2016, Съюз на учените в България – клон Пловдив, 28-29.10.2016, Plovdiv, Bulgaria

5. ***Adnan Redzheb***, Ognian Nakov, CONTROL METHODS FOR PARALLEL INSTRUCTIONS EXECUTION, **конференция** „ Дни на науката“ 2016, Съюз на учените в България – клон Пловдив, 28-29.10.2016, Plovdiv, Bulgaria

ИЗПОЛЗВАНИ ИЗТОЧНИЦИ

[1] BBC; Docker, Inc., "Enterprise CI at BBC News," in *DockerCon Europe 2014*, 2014.

[2] Cloud Bees, Inc., "The Business Value of Continuous Delivery," 2015. [Online]. Available: http://pages.cloudbees.com/rs/cloudbees/images/Business-Value-of-Continuous-Delivery.pdf. [Accessed 20 01 2016].

[3] Cloud Bees, Inc., "Meet Jenkins," [Online]. Available: https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins. [Accessed 29 01 2016].

[4] Cognizant Research Center, "Why On-Demand Provisioning Enables Tighter Alignment of Test and Production Environments," 2013.

[5] J. J. Dongarra, "Performance of Various Computers Using Standard Linear Algebra Software in a Fortran Environment," *ACM SIGARCH Computer Architecture News,* vol. 16, no. 1, pp. 47-69, March 1988.

[6] J. J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software," *ACM SIGARCH Computer Architecture News,* vol. 20, no. 3, pp. 22-44, June 1992.

[7] GitHub Inc., "GitHub Help," [Online]. Available: https://help.github.com/. [Accessed 29 01 2016].

[8] IBM Corp., "Accelerate your Software Delivery Lifecycle with IBM Development and Test Environment Services," IBM Corp., 2014.

[9] Intel Corp., "Linux Containers Streamline Virtualization and Complement Hypervisor-Based Virtual Machines," 2014.

[10] N. Khare, Docker Cookbook, Packt Publishing, 2015.

[11] Kubisys Inc., "The Better Way to Create Test Environments," 26 5 2015. [Online]. Available: http://www.kubisys.com/the-better-way-to-create-test-environments/. [Accessed 20 01 2016].

[12] F. McMahon, "The Livermore Fortran Kernels: A Computer Test Of The Numerical Performance Range," Lawrence Livermore National Laboratory, Livermore, CA, USA, 1986.

[13] Oracle Corp., "Oracle VirtualBox Documentation," [Online]. Available: https://www.virtualbox.org/wiki/Documentation. [Accessed 29 01 2016].

[14] Red Hat, Inc.; Cisco Systems, Inc., "Linux Containers: Why They're in Your Future and What Has to Happen First," 2014.

[15] S. Schneider, "How VMware IT Reduced Provisioning Time by 80%," 6 6 2012. [Online]. Available: https://blogs.vmware.com/vfabric/2012/06/how-vmware-it-reduced-provisioning-time-by-80-using-vfabric-application-director-and-more.html. [Accessed 20 01 2016].

[16] Sonatype, Inc., "Repository Management with Nexus," [Online]. Available: https://books.sonatype.com/nexus-book/reference/. [Accessed 29 01 2016].

[17] C. Strachey, "Time Sharing in Large Fast Computers," in *International Conference on Information Processing*, 1959.

[18] G. Tassey, "The Economic Impacts of Inadequate Infrastructure for Software Testing," National Institute of Standards and Technology, 2002.

[19] The Apache Software Foundation, "Welcome to Apache Maven," [Online]. Available: https://maven.apache.org/. [Accessed 29 01 2016].

[20] VMware, Inc, "Automated Software Development Life Cycle (SDLC) Provisioning on the VMware Private Cloud," 2013. [Online]. Available: https://www.vmware.com/files/pdf/VMware_IT_Automated_SDLC_Provisioning_Private_Cloud.pdf. [Accessed 20 01 2016].

[21] R. Weicker, "Dhrystone: A synthetic Systems Programming Benchmark," *Communications of the ACM,* vol. 27, no. 10, pp. 1013-1030, October 1984.

[22] B. A. Wichmann, H. J. Curnow and T. Si, "Whetstone: A Synthetic Benchmark," *The Computer Journal,* vol. 19, no. 1, pp. 43--49, 1976.

[23] K. Matthias and S. P. Kane, Docker: Up & Running, 1 ed., O'Reilly Media, 2015.

[24] S. Chacon and B. Straub, Pro Git, 2 ed., Apress, 2014.

[25] W. Gajda, Pro Vagrant, 1 ed., Apress, 2015.

[26] R. Black, E. v. Veenendaal and D. Graham, Foundations of Software Testing ISTQB Certification, 3 ed., Cengage Learning EMEA, 2012.

[27] R. Harrop and C. Ho, Pro Spring 3, 2 ed., Apress, 2012.

[28] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, 1 ed., Addison-Wesley Professional, 2010.

[29] D. J. Lilja, Measuring computer performance, A practitioner's guide, 1 ed., Cambridge, Great Britain: Cambridge University Press, 2000, p. 280.

[30] C. R. Johnson and P. Okunev, "Necessary And Sufficient Conditions For Existence of the LU Factorization of an Arbitrary Matrix," p. 21, 31 July 1997.

[31] E. Liu, "Conditioning and Numerical Stability," 2016. [Online]. Available: http://web.mit.edu/ehliu/Public/Yelp/conditioning_and_precision.pdf. [Accessed 25 2 2016].

[32] Intel Corp., "Intel Math Kernel - LINPACK," [Online]. Available: https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download. [Accessed 25 2 2016].

[33] P. J. Fleming and J. J. Wallace, "How not to lie with statistics: the correct way to summarize benchmark results," *Communications of the ACM - The MIT Press scientific computation series,* vol. 29, no. 3, pp. 218-221, March 1986.

[34] D. Salomon, G. Motta and D. Bryant, Handbook of Data Compression, 5 ed., Springer, 2010, p. 1383.

[35] M. Adler, "pigz - Parallel gzip," [Online]. Available: http://zlib.net/pigz/. [Accessed 26 2 2016].

[36] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory,* vol. 23, no. 3, pp. 337-343, May 1977.

[37] M. Mahoney, "Large Text Compression Benchmark," [Online]. Available: http://mattmahoney.net/dc/text.html. [Accessed 26 2 2016].

[38] J. M. Slocum, "Testing Memory I/O Bandwidth," 16 10 2013. [Online]. Available: http://jamesslocum.com/post/64209577678. [Accessed 26 2 2016].

[39] R. Longbottom, "Linux PC Benchmarks: RandMem Benchmark," [Online]. Available: http://www.roylongbottom.org.uk/linux%20benchmarks.htm#anchor9. [Accessed 26 2 2016].

[40] P. Dykstra, "nuttcp quick start guide," Distributed Computing IT Journal, May 2008. [Online]. Available: http://dcitj.blogspot.bg/2008/05/phil-dykstras-nuttcp-quick-start-guide.html. [Accessed 26 2 2016].

[41] HashiCorp, "Vagrant Private Networks," [Online]. Available: https://www.vagrantup.com/docs/networking/private_network.html. [Accessed 26 2 2016].

[42] Oracle Corp., "VirtualBox Virtual networking hardware," Oracle , [Online]. Available: https://www.virtualbox.org/manual/ch06.html#nichardware. [Accessed 26 2 2016].

[43] J. Gervais, "Smart NICs preserve CPU cycles," *Network World,* vol. 17, no. 50, pp. 69-70, 11 December 2000.

[44] StorageReview, "Fio - Flexible I/O Tester Synthetic Benchmark," [Online]. Available: http://www.storagereview.com/fio_flexible_i_o_tester_synthetic_benchmark. [Accessed 26 2 2016].

[45] Docker Inc., "Docker and OverlayFS in practice," [Online]. Available: https://docs.docker.com/engine/userguide/storagedriver/overlayfs-driver/. [Accessed 26 2 2016].

[46] K. Hess, "When Memory Serves You: Using ramfs and tmpfs," *Linix Magazine,* 31 January 2010.

[47] R. Arpaci-Dusseau and A. Arpaci-Dusseau, "Chapter: Faster Translations (TLBs)," in *Operating Systems: Three Easy Pieces*, Arpaci-Dusseau Books, 2014.

[48] R. Bhargava, B. Serebrin, F. Spadini and S. Manne, "Accelerating two-dimensional page walks for virtualized systems," *ACM,* pp. 26-35, 2008.

[49] I. Atkins, "Getting The Hang Of IOPS," 28 June 2012. [Online]. Available: http://www.symantec.com/connect/articles/getting-hang-iops-v13. [Accessed 27 2 2016].

[50] M. Larabel, "Ubuntu 15.10: KVM vs. Xen vs. VirtualBox Virtualization Performance," 22 10 2015. [Online]. Available: https://www.phoronix.com/scan.php?page=article&item=ubuntu-1510-virt&num=2. [Accessed 27 2 2016].

[51] Wikipedia, "IOPS," 12 February 2016. [Online]. Available: https://en.wikipedia.org/wiki/IOPS. [Accessed 27 2 2016].

[52] Linux community, "Linux manual for fio," [Online]. Available: http://linux.die.net/man/1/fio. [Accessed 26 2 2016].

[53] Linux community, "Linux manual for fsync," [Online]. Available: http://linux.die.net/man/2/fsync. [Accessed 27 2 2016].

[54] Cannonical, "Ubuntu manual for mbw," Cannonical, [Online]. Available: http://manpages.ubuntu.com/manpages/utopic/man1/mbw.1.html. [Accessed 26 2 2016].

[55] Cannonical, "Ubuntu manual for nuttcp," Cannonical, [Online]. Available: http://nuttcp.net/nuttcp/nuttcp.html. [Accessed 26 2 2016].

[56] Cannonical, "Ubuntu manual for netperf," Cannonical, [Online]. Available: http://manpages.ubuntu.com/manpages/natty/man1/netperf.1.html. [Accessed 26 2 2016].

[57] Linux community, "Networking - ip-sysct," [Online]. Available: https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt. [Accessed 27 2 2016].

[58] S. Haines, "Continuous Integration and Performance Testing," *Dr. Dobb's The Worls of Software Development,* p. 3, 7 February 2008.

[59] C. Sauer, A. Gemino and B. H. Reich, "The impact of size and volatility on IT project performance," *Communications of the ACM,* vol. 50, no. 11, pp. 79-84, November 2007.

[60] Standish Group, "Chaos Report," Standish Group, 2015.

[61] M. Poppendieck and T. Poppendieck, Implementing Lean Software Development, 1 ed., Addison-Wesley Professional, 2006.

[62] M. Poppendieck and T. Tom Poppendieck, Lean Software Development: An Agile Toolkit, Addison-Wesley Professional, 2003.

[63] K. Ellis, "The Impact of Business Requirements on the Success of Technology Projects," IAG Consulting, 2009.

[64] TEM, "Estimating the Speed of Exponential Technological Advancement," 2012.

[65] eBay PaaS Team, "Delivering eBay's CI Solution with Apache Mesos," eBay, 2014.

[66] C. Coleman, "The Future of OpenShift and Docker Containers," Red Hat, 2014.

[67] Docker, Inc., "Docker Compose," [Online]. Available: https://docs.docker.com/compose/. [Accessed 27 2 2016].

[68] Intel Corp., "The Advantages of Using Virtualization Technology in the Enterprise," 5 March 2012. [Online]. Available: https://software.intel.com/en-us/articles/the-advantages-of-using-virtualization-technology-in-the-enterprise. [Accessed 28 2 2016].

[69] VMWare Inc., "Accelerate Software Development Testing and Deployment with VMWare Virtualization Platform," VMWare Inc., 2005.

[70] B. A. Wichmann, A. A. Canning, D. L. Clutterbuck, L. A. Winsborrow, N. J. Ward and D. W. R. Marsh, "Industrial perspective on static analysis," *Software Engineering Journal,* vol. 10, no. 2, pp. 69 - 75, March 1995.

[71] G. J. Myers, C. Sandle and T. Badgett, The Art of Software Testing, 3rd ed., Wiley, 2011, p. 256.

[72] H. D. Benington, "Production of Large Computer Programs," *Annals of the History of Computing,* vol. 5, no. 4, pp. 350-361, November 1983.

[73] J. M. Wilson, "Gantt charts: A centenary appreciation," *European Journal of Operational Research,* vol. 149, no. 2, p. 430–437, September 2003.

[74] W. R. Franta and K. Maly, "An efficient data structure for the simulation event set," *Communications of the ACM,* vol. 20, no. 8, pp. 596-602, August 1977.

[75] C. Pautasso, O. Zimmermann and F. Leymann, "RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision," in *17th International World Wide Web Conference (WWW2008)*, Beijing, China, 2008.

[76] D. J. Panzl, "Test procedures: A new approach to software verification," in *ICSE '76 Proceedings of the 2nd international conference on Software engineering*, Los Alamitos, CA, USA, 1976.

[77] R. V. Binder, Testing Object-Oriented Systems: Models, Patterns, and Tools, Addison-Wesley Professional, 1999.

[78] IEEE Computer Society, IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries, IEEE, 1991, pp. 1 - 217.

[79] B. Beizer, Software system testing and quality assurance, New York, NY, USA: Van Nostrand Reinhold Co., 1984.

[80] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, "Xen and the art of virtualization," in *SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles*, New York, NY, USA, 2003.

[81] R. Dua, A. R. Raja and D. Kakadia, "Virtualization vs Containerization to Support PaaS," in *2014 IEEE International Conference on Cloud Engineering (IC2E)*, Boston, MA, USA, 2014.

[82] N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *IEEE Communications Magazine,* vol. 47, no. 7, pp. 20 - 26, July 2009.

[83] D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment," *Linux Journal,* vol. 2014, no. 239, March 2014.

[84] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing,* vol. 1, no. 3, pp. 81-84, September 2014.

[85] A. J. Younge, R. Henschel, J. T. Brown, G. v. Laszewski, J. Qiu and G. C. Fox, "Analysis of Virtualization Technologies for High Performance Computing Environments," in *2011 IEEE International Conference on Cloud Computing (CLOUD)*, Washington, DC, USA, 2011.

[86] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer and V. Paxson, "The Matter of Heartbleed," in *IMC '14 Proceedings of the 2014 Conference on Internet Measurement Conference*, New York, NY, USA, 2014.

[87] IEEE Computer Society, IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture, IEEE, 202, pp. 1 - 48.

[88] P. Li, "Selecting and using virtualization solutions: our experiences with VMware and VirtualBox," *Journal of Computing Sciences in Colleges,* vol. 25, no. 3, pp. 11-17, January 2010.

[89] J. Watson, "VirtualBox: bits and bytes masquerading as machines," *Linux Journa,* vol. 2008, no. 166, p. Article No. 1, February 2008.

[90] J. Palat, "Introducing vagrant," *Linux Journal,* vol. 2012, no. 220, p. Article No. 2, August 2012.

[91] R. Rosen, "Resource management: Linux kernel Namespaces and cgroups," University of Santa Barbara, Santa Barbara, CA, USA, 2013.

[92] J. Petazzoni, "Anatomy of a Container: Namespaces, cgroups & Some Filesystem Magic," in *LinuxCon North America*, Seattle, WA, USA, 2015.

[93] S. Yegulalp, "CoreOS launches Rocket 1.0 directly at Docker," 5 February 2016. [Online]. Available: http://www.infoworld.com/article/3029682/application-virtualization/coreos-launches-rocket-10-directly-at-docker.html. [Accessed 3 3 2016].

[94] B. Ballard, "Docker: The business benefits," 4 December 2015. [Online]. Available: http://betanews.com/2015/12/10/docker-the-business-benefits/. [Accessed 2 3 2016].

[95] M. Bauer, "Paranoid penguin: an introduction to Novell AppArmor," *Linux Journal,* vol. 2006, no. 148, p. 13, August 2006.

[96] B. McCarty, SELinux: NSA's Open Source Security Enhanced Linux, O'Reilly, 2004.

[97] J. S. Heidemann and G. J. Popek, "File-system development with stackable layers," *ACM Transactions on Computer Systems (TOCS) - Special issue on operating systems principles,* vol. 12, no. 1, pp. 58-89, February 1994.

[98] X. Wu, W. Wang and S. Jiang, "TotalCOW: Unleash the Power of Copy-On-Write for Thin-provisioned Containers," in *6th Asia-Pacific Workshop on Systems*, Tokyo, Japan, 2015.

[99] F. Kereki, "Concerning containers' connections: on docker networking," *Linux Journal,* vol. 2015, no. 254, p. Article No. 2, June 2015.

[100] P. Hunt, M. Konar, F. P. Junqueira and B. Reed, "ZooKeeper: Wait-free coordination for Internet-scale systems," in *USENIX Annual Technical Conference*, Boston, MA, USA, 2010.

[101] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, 1st ed., Addison-Wesley Professional, 1994, pp. 1-395.

[102] R. Mason, "Load Balancing Apache Tomcat with Nginx," 3 September 2013. [Online]. Available: https://dzone.com/articles/load-balancing-apache-tomcat. [Accessed 3 3 2016].

[103] D. Ståhla and J. Boschb, "Modeling continuous integration practice differences in industry software development," *Journal of Systems and Software,* vol. 87, no. 1, p. 48–59, 2014.

[104] C. Ramey, "Bash, the Bourne-Again Shell," Case Western Reserve University, Cleveland, OH, USA, 1994.

[105] VMWare Inc., "Virtualization Overview," VMWare Inc., Palo Alto, CA, USA, 2006.

[106] E. W. Biederman, "Multiple instances of the global Linux namespaces," in *Ottawa Linux Symposium*, Ottawa, Canada, 2006.

[107] R. Shea and J. Liu, "Understanding the impact of Denial of Service attacks on Virtual Machines," in *IEEE 20th International Workshop on Quality of Service, IWQoS '12*, Coimbra, Portugal, 2012.

[108] M. G. Xavier, "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments," in *21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, Belfast, Ireland, 2013.

[109] V. Venkatesh, M. G. Morris, G. B. Davis and F. D. Davis, "User Acceptance of Information Technology: Toward a Unified View," *MIS Quarterly,* vol. 27, no. 3, p. 425–78, September 2003.

ABSTRACT of Ph.D. THESIS

This dissertation examines optimization methods for parallel processing of information. It analyzes the methods already known and presents different implementation methods suitable for analysis and comparison with the new processing methods, which are proposed in the dissertation. An environment for performing the necessary measurements of parallel processing is developed. A new method implemented through framework is proposed. Such framework contains classes that demonstrate a new arrangement of the task sequence, based on queues with different priorities. The priority is determined by a different criterion and the scanning of the queues is based on a circular order (Round Robin). The tasks are arranged before entering the thread pool and test measurements are being made. The results show performance acceleration in comparison to the known methods and the direct access to the thread pool. Despite the resources necessary for rearrangement, the final result is that the execution time is reduced. The result of the proposed method is positive.

The contributions of the dissertation can be summarized as follows:

- Detailed analysis and synthesis of basic classical algorithms for parallelizing processes and tasks;
- Analysis of algorithms to scan structures (task containers);
- Proposed new method upgrading the .NET ThreadPool on the basis of performed analyzes;
- Comparison between the classical methods and the method designed by the PhD candidate;
- Proving an increased productivity.

In his thesis the PhD candidate presents an in-depth knowledge in parallel information processing, high level framework implementation, and analysis of algorithms for scanning of various structures