TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Institute of Computer Science

ITT70LT

Roman Školin, 132429IAPM

PARALLEL AND DISTRIBUTED COMPUTING APPROACH FOR FINANCIAL DERIVATIVES PRICING

Master thesis

Advisor: Pavel Grigorenko

Doctor of Philosophy

Researcher

Tallinn

2016

I declare that this thesis is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature	
Name	Roman Školin
Date	May 9, 2016

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Arvutiteaduse instituut

ITT70LT

Roman Školin, 132429IAPM

PARALLEEL- JA HAJUSARVUTUSTE KASUTAMINE TULETISLEPINGUTE HINDAMISEL

Magistritöö

Juhendaja: Pavel Grigorenko

Doktorikraad

Teadur

Tallinn

2016

ABSTRACT

Derivative contracts are a kind of financial instruments which play an important role in the financial markets. The complexity of derivatives, as well as the need for the price forecasting and accurate analysis of forecast results represent the main problems the derivatives market is facing. Parallel and Distributed computing, special techniques and software packages provide effective approaches to meet these challenges. The goal of this work is to conduct the study of methods for solving the problem of financial derivatives pricing. The project is motivated by long-lasting calculations of the financial forecasts, providing inaccurate results and costing a full working day in case of the need for the recalculation.

Theoretical part of the work addresses the set of financial methods used for derivatives pricing with respect to underlying asset price changes along with the description of the application of parallel and distributed computing techniques. The approach chosen for the thesis for solving the problem of long-lasting calculations is a distributed computational cluster. The experimental part contains the implementation of the prototype and evaluation of the selected method, taking into account functional and non-functional metrics – performance, set up effort, usage and maintenance costs. A result of this work is a high-level description of the distributed system capable of performing seamless calculations based on a given time series and model.

The thesis is in English and contains 84 pages of text, 6 chapters, 35 figures, 17 tables.

Annotatsioon

Tuletislepingud on finantsinstrumendid, mis mängivad olulist rolli finantsturgudel. Tuletislepingute keerukus, vajadus teostada kiireid hinnaprognoose ning täpseid prognoostulemite hindamisi, on üks peamiseid väljakutseid tuletisinstrumentide turul. Paralleel- ning hajusarvutuste rakendamine koos spetsiaalsete tehnikate ning tarkvaraga on antud probleemile efektiivseks lahenduseks. Käesolev magistritöö käsitleb tuletisinstrumentide hindade prognoosi probleemi, mille lahenduse teostus on inspireeritud pikalt kestvatest ning mittetäpseid tulemeid pakkuvatest finantsprognooside arvutustest, kus vea hind on võrdne ühe päeva ümberarvutustega. Töö teoreetiline osa kirjeldab mõningaid tuletislepingute hinnaprognooside arvutusmeetodeid, sobiliku meetodi valimist ning paralleel- ja hajusarvutustehnikate rakendamist. Lisaks eelnevale on kirjeldatud ühe väljavalitud meetodi põhjal olukorra parendamiseks pika kestvusega kalkulatsioonide üleandmist käepärastest vahenditest koostatud klastrile. Töö praktiline osa sisaldab süsteemi prototüübi dokumentatsiooni ning valmis lahendust prototüübi näol, mis teostab järgnevaid tegevusi:

- tulemuste tootlikkuse hindamine.
- kasutus- ning hoolduskulu vaatenurgast ressursinõudlikkuse hindamine.

Selle projekti tulemiks on abstraktse matemaatilise mudeli ning aegridade põhjal töötava hajussüsteemi kirjeldus.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 84 leheküljel, 6 peatükki, 35 joonist, 17 tabelit.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

Cloud the practice of using a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server or a personal computer. 9

CPU A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program. 21

Financial Derivative A derivative is a security with a price that is dependent upon or derived from one or more underlying assets. 9

FPGA A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing. 43

GPU A graphics processing unit (GPU) is a computer chip that performs rapid mathematical calculations, primarily for the purpose of rendering images. 9

MPI Message Passing Interface (MPI) is a standardized and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computers. 26

Option An option is a contract that gives the buyer the right, but not the obligation, to buy or sell an underlying asset at a specific price on or before a certain date. 9

ORM Object-relational mapping (ORM, O/RM, and O/R mapping tool) technique for converting data between incompatible type systems in object-oriented programming languages. 65

SQL Structured Query Language, special-purpose programming language designed for managing data held in a relational database. 65

VM A virtual machine (VM) is a software computer that, like a physical computer, runs an operating system and applications. 79

1. INTRODUCTION

This chapter provides an information about the financial derivative contracts, addresses the pricing problem and sets the goals of the current work.

1.1 Background and motivation

By definition Financial Derivative is a contract with a price that is derived from one or more underlying assets [1]. Any product or service can act as underlying asset. Derivative value changes follow changes in a price of the underlying assets. Financial Derivatives markets include manipulating the high stakes, so even the slightest error in the assessment of the contract may lead to large losses. As the nature of the financial markets is very unstable, it is important to estimate the possible changes in prices of underlying assets quickly and accurately. That requires solving complex computational tasks such as identification and verification of economic models, analytical and forward-looking estimates of price changes, creation of analytical materials on the basis of the forecast results.

Fair valuation of derivatives contracts with respect to the underlying assets price changes is called the *pricing problem.* Existing mathematical methods such as Black-Scholes equations [2] or Rubinstein-Cox-Ross binomial method [3] created a basis for derivatives pricing using analytical approach. The rapid development of computer technology has led to the possibility of applying the numerical methods for overcoming the limitations of analytical methods. As the numeric methods require large number of computations, parallel and distributed computing are the keywords to success.

Derivative pricing starts from the underlying asset price forecasting. On the basis of the price time series different existing systems try to calculate future prices. To illustrate the scope of computation – calculation of one Option contract containing 5 underlying assets with prices for last 10 years requires $7 * 10^6$ operations [4]. Cloud computing, high performance computing clusters, GPU-powered computations – all thise options are available for solving given complex computational problem. Several researches were done in the field of solving derivatives pricing problem, for example Hans Moritsch in his phd dissertation described computational problems in finance and proposed a mixture of Monte-Carlo method with backward induction, used in binomial method [5] [6]. In addition to distributing the computations between different resources, a remarkable parallelization of existing algorithms has been achieved. New trends involve distributed systems, built on the voluntary basis (Crowd computing). All the facts mentioned above are encouraging to investigate how is it possible to benefit from knowing the latest trends in parallel and distributed computing.

1.2 Goals

The major goal of this work is to research different ways of applying parallel and distributed techniques for solving the derivative pricing problem on the example of Option price calculations and to experiment with the approach, which provides the best result in terms of simplicity of implementation and trustworthiness of a results. The main objectives of this work are:

Theoretical objectives

- Understand the nature of financial derivatives in general and option pricing in particular.
- Understand the basics of stochastic model based option price valuation methods.
- Research particular cases of applying parallel and distributed computing methods for option pricing.
- Research general parallel and distributed computing patterns and Map and Reduce primitives in particular.
- On the basis of classified information to design a system, capable of forecasting option underlying assets prices using the methodologies of parallel and distributed computing in combination with Crowd computing approach.

Practical objectives

- Implement a prototype of the designed system
- Evaluate the performance of the designed prototype, bring out benefits and bottlenecks.

1.3 The Problem Statement

Maintaining large data sets, costly long-lasting calculations using complicated mathematical models and high rental fees for dedicated super computing services is a serious problem that the companies dealing with financial analysis on the daily basis are facing. Some fortunate companies are able to use out of the box solutions, while others utilize in-house modeling tools, private historical data and servers working at the edge of capacity.

There are many approaches, which could help to solve this problem. Companies can buy more powerful servers or optimize the software or review the practices of working with databases, change to faster hard disks or move computing to a cloud platform and use the services, which contain a powerful distributed computing system under the hood. One side effect of all of these approaches is a significant cost.

The problem can be described as a difficult choice between modern technical means and principles and selection the best option in terms of a limited budget, which would provide stable and reliable result of large-scale calculations along with acceptable calculation speed.

1.4 Outline of the thesis

The thesis is organized as follows: the second chapter "Theoretical overview" introduces existing approaches of the option pricing, parallel and distributed computing techniques and gives insights for choosing the methodology to perform option pricing calculations in accordance to the goals of the work.

The "Related work" chapter describes current state of art in the financial derivatives pricing field and gives a brief overview of the financial derivatives valuation with underlying assets price forecasting using numerical methods and pricing models. Monte-Carlo approach is brought up. Some systems for model-based price forecasting are described. The chapter contains a description of existing generic parallel and distributed computing frameworks and systems. Creating of the computing cluster, which utilizes MapReduce pattern and Crowd computing principles, is brought out as a possible solution to the stated problem.

The chapter "System Design And Implementation" briefly describes the results of the created cluster testing. During the testing of the whole system Monte-Carlo simulation of Option price forecast calculation was done in parallel using several available computers. As an input for the testing 2000 assets were used. Each asset contains price time series from year 1960, which makes 20440 days by the moment of writing. The chapters contains some notes about system benefits, weak sides and technical constraints.

The last chapter contain conclusions and ideas for the future work.

2. THEORETICAL OVERVIEW

This chapter describes the existing approaches of the option pricing, parallel and distributed computing techniques and gives and insights for choosing of the methodology to perform option pricing calculations in accordance to the goals of the work.

2.1 Option Pricing

An option is one of the main derivatives used for reducing risks at the financial markets. By definition option is a contract that gives buyer a right, but not the obligation to buy or sell an underlying asset at a specific price on or before a certain date. An option, just like a stock or bond, is a security. It is also a binding contract with strictly defined terms and properties [4]. Depending on the option style it can have different execution times, for example, an American option can be executed at any time before some defined period and European option executes only at the defined moment.

Option pricing is based on the underlying asset price forecast. Two main approaches can be distinguished. First, the analytical approach, relies on the stochastic equation and, second, the simulations approach, bases on numerical methods and a large amount of calculations.

Existing Mehtods

There are 2 types of options - *Call* [7] and *Put* [8].

Call – option gives an option buyer the right to buy the underlying asset from a seller of the option at the *Strikeprice* on time or abandon the purchase.

Put – option gives an option buyer the right to sell the underlying asset at the *Strikeprice* on time or the seller of the option to abandon the sale.

Strikeprice – The price at which a contract can be exercised [9]

The value of possible revenue from the buying or selling option is the subject to calculate in this work. Several existing well-known numerical and analytical methods of have been considered a starting point to explore the possibilities of applying the techniques of parallel and distributed computing for Option price forecasting. The observed methodologies are:

- Black-Scholes model [2]
- Binomial method [3]
- Monte-Carlo Simulation [10]

All evaluation methods are based on the assumption of a possible change in the price of the underlying asset. Certain probability weight is assigned for an each possible price change for a given interval. The changes in price of the underlying asset are assumed to be have a log-normal distribution. Return on assets is equal to the relative change in price of the underlying assets during the option trading period.

The initial parameters for all three methods are the price volatility σ, measured as the standard deviation of the yield on the asset, the average rate of change of the price of the asset μ, expressed as a simple annual interest and calculated as the expectation of return in one step τ, the Strike price K, lifetime T, Spot price (current underlying asset price) S at time moment $t = 0$, and risk-free rate r, according to which income is discounted from assets

Black-Scholes. Option pricing model was first published in 1973, in an article entitled "Pricing of Options and Corporate Liabilities" in the journal "Political Economy". Black-Scholes model is used to calculate the possible price for Put and $Call$ options.

This model relies on several beliefs and assumptions:

- The right to buy or sell can be used only on the day of the option expiration
- Dividends on assets, participating in options, are not paid during option lifetime
- Market fluctuations can not be predicted
- Commissions and transaction costs are ignored
- Risk-free interest rate [11] and the volatility of the relevant asset options are known constants.
- Option underlying assets' price and revenue changes are distributed normally

The Black-Scholes pricing equation for call options:

$$C = NS(d_1) - N(d_2)Ke^{-rt} \tag{2.1}$$

$$d_1 = \frac{\ln(\frac{S}{K}) + (r + \frac{s^2}{2})}{s\sqrt{t}} \tag{2.2}$$

$$d_2 = d_1 - s\sqrt{t} \tag{2.3}$$

where:

C = Option revenue

S = Current underlying asset price

t = Time for option exercise

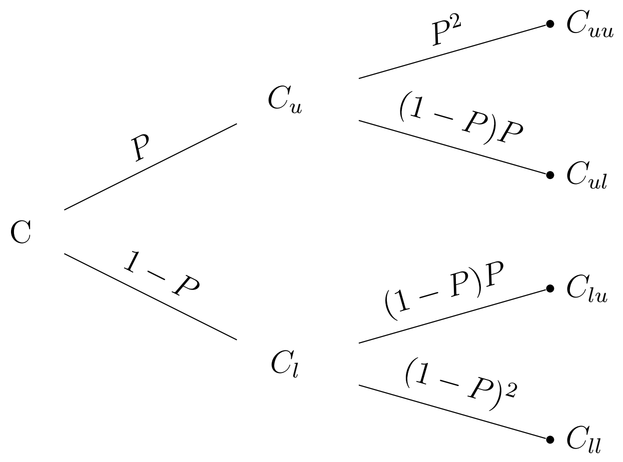
K = Strikeprice [9]

r = Risk-free intrest rate [11]

N = Normal distribution of prices (taken from tables)

s = Normal distribution of underlying asset prices aka volatility (taken from tables)

The Binomial method. This method was developed in 1979 and is based on constructing of a pricing binary tree in which is based on the assumption that in the future, after a certain period of time the price of the asset will rise by some certain value with a probability P or decreases by a certain value with probability $1 - P$.



Asset price after taken period is calculated using equation 2.4

$$C = \frac{PC_u + (1-P)C_l}{r} \tag{2.4}$$

where C_u is the possible highest price at after the taken period time interval and C_l is the lowest price after the taken time interval. Time intervals can be added by building new branches of a binary tree. The amount of revenue is the weighted price of the asset. The method can be implemented as a direct construction of the tree recursively. With that particular case parallel execution can be used in case if the possible C_u and C_l is known and appropriate pair of them is passed to every correspondent recursion.

Monte-Carlo simulation. Monte-Carlo simulation is based on constructing a sufficiently large number of calculation iterations and averaging the calculation results. Monte-Carlo method is used when an additional level of input data flexibility is needed (analysts want to bring in additional fluctuations in the input parameters or the parameter list includes multiple uncertainties). Monte-Carlo simulation flow is following:

- A set of underlying asset prices is calculated using equation

$$S_t = S_0\, e^{(r-\frac{\sigma^2}{2})+\sigma B(t)} \tag{2.5}$$

- When the set of asset prices is generated, the option price is calculated. Given equation demonstrates the calculation of the $Call$ option price.

$$C_t = e^{-r(t)} E\, max[(0, S_t - X)] \tag{2.6}$$

- After calculation of the option price series, the mean price is calculated

$$C_{mean} = \frac{1}{n} \sum_{i=1}^{N} C_i(S,T) \tag{2.7}$$

The input date for the equations is given as following:

$S(T)$ = Underlying asset price by the end of calculation period

$S(0)$ = Current underlying asset price

t = Expiration time

σ = volatility

r = Risk-free intrest rate [11].

X = Strikeprice [9]

B_t = Brownian motion

Mean value of the calculation is the value of the possible option price.

2.1.1 Parallel and Distributed Computing Ideas

In this work Option premium calculation will be observed only in the context of applying of parallelism and distribution for pricing problem solving. On simple example of distributed computations is calculation of the stock assets price forecast using several physical or virtual machines which are orchestrated in some particular manner. The main focus is set to the investigation of the ways to perform distributed calculations using the standard computers – regular PCs or Notebooks with most common characteristics and without any specific abilities to perform high-performance operations with a large sets of input data.

The Black-Scholes model is the most favored method of the option premium valuation and it is quite accurate. It relies on solid data, such as current stock prices, strike prices, expiration time, risk-free rate, volatility, absence of dividends and commission and transaction fees. For most of the cases Black-Scholes model is good enough. The disadvantage of the Black-Scholes model is inflexibility, which makes it rather difficult to use with non-typical features, for example price reset aspect or obligation of mandatory exercise. In terms of digital prototyping, no big dataset processing is needed, all the data required for Option price forecast can be scouted from the exchange tables. This method is also known as analytical method, thus, is presented just declaratively and not to be implemented in prototyping section.

In comparison to the Black-Scholes, a Binomial method splits the life-cycle of the option to the time of expiration into several steps. At each step two moves for the stock price are possible – one up and one down. This produces a binomial distribution of underlying asset prices. As a result, the model creates a theoretical representation of the all possible prices, which stock prices could take over option life cycle. This model is more flexible than Black-Scholes as there is a possibility to change input parameters at a randomly selected time moment. Technically that method can be implemented using recursive calls of price calculation kernel, but in terms of parallel and distributed computing it is hard to create independent recursive calls in parallel.A need for backward propagation of the calculated values from iteration to iteration makes paralellizing of that method way too hard. Another aspect is the need to evaluate a complexity of the algorithm before implementation, which is also quite difficult for the parallel recursion.

Monte-Carlo method uses a multiple calculations to predict outcomes. Selected model is just calculated over and over again providing a new Option price at each iteration. After the sufficient amount of calculations is done, the average of the all results is returned as a possible Option price. Monte-Carlo method provides the best of both methods described above: it provides an additional flexibility in comparison to Black-Scholes model, allowing user to drop in some input changes on the fly, and it is pretty easy to evaluate the algorithmic part and implement it in code. It is also the best candidate to be run in parallel and to be distributed over several computers. With such advantages, Monte-Carlo simulation is the best candidate for prototyping part of this work.

2.2 Parallel Computing

2.2.1 Definition of Parallel Computing

Parallel computing is a way where computational work is done by several processors simultaneously. Parallel computing solves the problems of expensive computations by dividing the work between several processing units. Nowadays all supercomputers and most of the existing consumer computers utilize parallel computing principles. For composing of this particular section Pace University's resources [12] were used.

2.2.2 Why use Parallel Computing?

Parallel computing is a key to full utilization of modern multi-core processors. There are a lot of areas, where traditional serial approach is outdated and parallel computations are the best match. One of such areas is scientific modeling – from weather forecast to planetary movements. Parallel computing saves time and money by accomplishing given tasks faster. It can solve tasks, which require so much resources that a single computer might not have enough. [13]

From the software development perspective, the spreading of multi-core processors already caused revolutionary changes. The majority of modern programs are written as parallel, and the ones which are still sequential, are going to be either converted or abandoned. The main reasons for that are given below :

- For the efficient processor use it is very important to keep it busy, and the only way to keep busy a multi-core processor is to make a parallel program, meaning that the program will run on all processor cores.

- Next reason it also the consequence of processor architecture changes – sequential code, written for one CPU will run slower because of multi-core CPU clock rate decline in favour of adding more cores.
- It is also worth of noticing the constantly growing gap between the actual speed of processors and memory latency (the situation when some variable delivery from the memory to processor takes much longer than a time, which is needed for a processing command execution)

2.2.3 Parallel Architectures

There is a wide variety of different parallel architectures and it is very difficult to create a fair classification of parallel architecture types. Although different types of parallel architecture have overlapping characteristics, it is still possible to distinguish different parallel architectures into several groups:

- Flynn's taxonomy
- Instruction- and data stream based classification
- Classification based on the type of processing elements
- Specific Parallel Architectures

That division is not completely fair as different sources and different materials can give different versions of classification with further explanations.

Flynn's taxonomy

One of the most widely used classifications since 1966, which distinguishes multi-processor computer architecture types according to how they can be classified along the two dimensions – Instruction Stream and Data Stream.

Each of these dimensions can have only one of two possible states: Single or Multiple. [14].

SISD - Single Instruction, Single Data

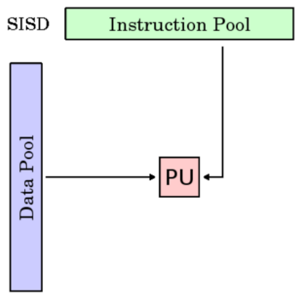


Fig. 2.1: SISD system

- A serial computer
- Single Instruction: Only one instruction stream is being acted on by the CPU during each clock cycle
- Single Data: Only one data stream is being used as input during each clock cycle
- Deterministic execution
- This is the oldest type of computer
  Examples: older generation mainframes, minicomputers, workstations and single processor/core PCs.

SIMD - Single Instruction, Multiple Data

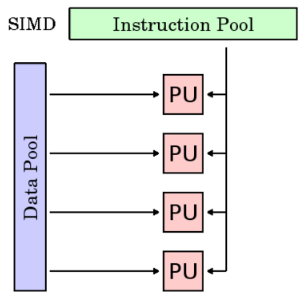


Fig. 2.2: SIMD system

- A type of parallel computer
- Single Instruction: All processing units execute the same instruction at any given clock cycle
- Multiple Data: Each processing unit can operate on a different data element
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Synchronous and deterministic execution
- Processor Arrays and Vector Pipelines
- Examples:
  Processor Arrays: Thinking Machines CM-2, MasPar MP-1 and MP-2, IlliAC IV Vector
  Pipelines: IBM 9000, Cray X-MP, Y-MP and C90, Fujitsu VP, NEC SX-2, HItachi S820, ETA10

- Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.

MISD - Multiple Instruction, Single Data

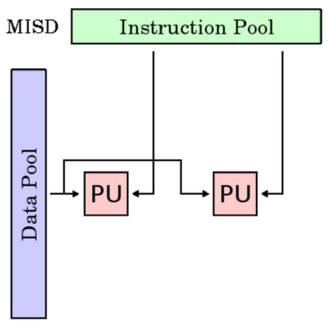


Fig. 2.3: MISD system

- A type of parallel computer
- Multiple Instruction: Each processing unit operates on the data independently via separate instruction streams.
- Single Data: A single data stream is fed into multiple processing units.
- Few actual examples of this class of parallel computer have ever existed.
- Some conceivable uses might be: multiple frequency filters operating on a single signal stream, multiple cryptography algorithms attempting to crack a single coded message.

MIMD - Multiple Instruction, Multiple Data

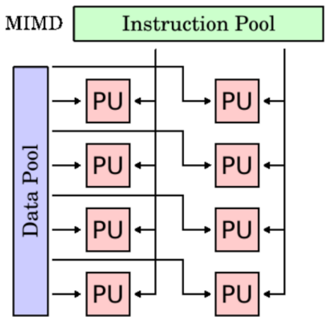


Fig. 2.4: MIMD system

- A type of parallel computer
- Multiple Instruction: Every processor may be executing a different instruction stream
- Multiple Data: Every processor may be working with a different data stream.
- Execution can be synchronous or asynchronous, deterministic or non-deterministic.
- Currently, the most common type of parallel computer - most modern supercomputers fall into this category.
- Examples: most current supercomputers, networked parallel computer clusters and grids, multi-processor SMP computers, multi-core PCs.

Instruction- and data stream based classification

In terms of memory arrangement Parallel architectures can be divided into 2 major categories. These are: shared memory and message passing or distributed memory. Shared memory and distributed memory architectures are also called tightly coupled and loosely coupled architectures respectively. [15]

Shared Memory Model

In accordance to the Parallel Architectures Classifications by Springer [16] In this model multiple processors share a common memory unit comprising a single or several memory modules. Processors of the systems are communicating with each other by writing information into common memory and reading it out.

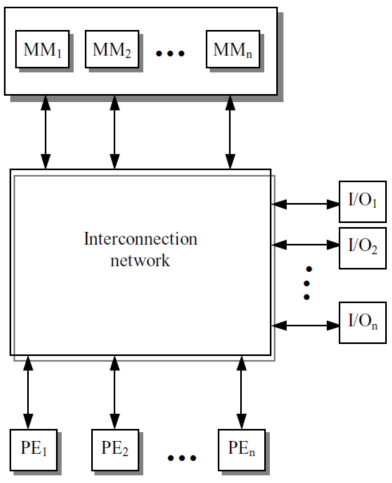


Fig. 2.5: Shared memory architecture

Message Passing Model

Is different from the Shared Memory architecture in a way, that each unit in this model is a standalone computer with it's own memory, processor and IO devices. Communication among the processors is established in the form of I/O operations through message signals and network bus. For example, if a processor needs data from another processor it sends a signal to that processor through an interconnected bus network demanding the required data. The remote processor then responds accordingly. This kind of system is also known as distributed memory system.

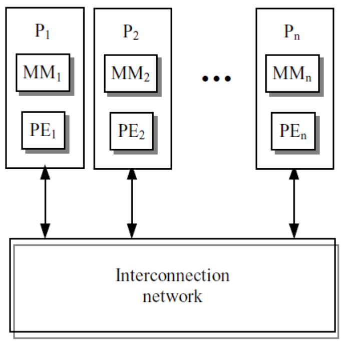


Fig. 2.6: Distributed memory architecture

Classification based on the type of processing elements

RISC

Reduced Instruction Set Computer – processing element, oriented to perform a limited instruction set in a vary fast and effective way. [17]

CISC

Complex Instruction Set Computer – processing element capable of carrying out many tasks such as memory IO and mathematical operations. [18]

Vector processors

Vector processors are designed to execute numerous matrix operations in a very efficient way. [19]

Specific types of parallel architectures

Specific parallel architectures are architectures, which do not follow any common pattern, may vary from system to system and are meant to perform specific tasks. One example of the specific parallelism is a pipelining – the way of executing by breaking tasks into different processes and execution of these processes on different processing units. Each computation unit within a pipeline is responsible for special operation.

2.2.4 Parallel Programming Models

Parallel programming models are abstractions above Parallel memory and hardware architectures. This work will bring out several most well-known and widely used models:

- Message Passing
- Data Parallelism
- Shared memory

Message Passing

In this model programs are sharing data using transmission and reception from process to process through cooperative operations of communication system. Aggregating and standardization of the various messaging libraries let to the foundation of MPI [20] standard.

Data Parallelism

In this model the main program determines how data is distributed among all of the processors in the system. All operations over each element of the data set are made in parallel, for that reason data set is being partitioned in a way, which allows different threads to work with different data sections simultaneously. Computational data can be split into different chunks and every chunk is processed by separated computation node. The term “node” here is declarative and can present a thread in multi-threaded program or a machine in computation cluster. No dependencies between processing different chunks of the data makes that model perfect in terms of concurrency.

Shared memory

In this model all processes share a common memory address space. That creates an overhead when choosing a moment when you can put the data into memory and when it is possible to remove it. Control over access to shared memory is done using standard synchronization mechanisms – semaphores and locking processes. The example of a shared memory model a threading model with explicit thread management and control over accessed memory. This approach is more complicated than data parallelism due to the need to take into account such constraint that a single piece of memory space can not be used in computation simultaneously.

Parallel Application Development

Development of parallel applications is still a complicated process because of several issues. At one hand there is a huge legacy of sequential programs, which work quite well and the overhead of parallel programming forces software developers to prefer writing of the sequential code everywhere, where deterministic execution result is expected.

Main challenges of the development in parallel are:

- Non-deterministic execution (hard to debug)
- Overhead of load-balancing (Difficulties with threads orchestration between processors in the system, as a result applying all threads to the one system core)

- Deadlocks and Race conditions – parallel code failures due to poor coding practises
- Synchronization and locking need

In general there are 2 approaches for parallel programming:

- Implicit parallelism – with this approach parallelizing is carried out by compilers or specific libraries (example: Parallel loops in .NET or Java 8). This is a way of dynamic parallelism, where the level of parallelism is decided without participation of a software developer.
- Explicit parallelism – in this approach creation and orchestrating of the parallelism and a level of decomposition, mapping to different processor cores, communication between threads, is done by a software developer. Also known as static parallelism which is more tricky to implement.

In the most cases it is a matter of choice to develop a new application from the scratch or to rewrite existing sequential application into parallel mode.

Algorithms

Parallel algorithms, in contrary to traditional serial algorithms, are the algorithm designed in a such way that they can be executed in parts on a variety of different computing devices and, combining the execution results, produce the correct result. Parallel algorithms can be represented as a directed acyclic graph where each vertex is a computational action (kernel invocation, reading, writing, etc). Directed edges show, how one computational action depends on another.

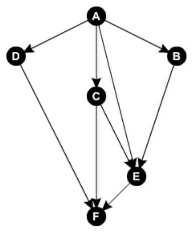


Fig. 2.7: Computation as graph

There are two main parameters for evaluation of the algorithm cost.

- Work complexity $W(n)$ - the amount of the vertices on the computational graph
- Step complexity $D(n)$ - the amount of the vertices on the longest path of the graph (Critical path through the graph)

An efficient parallel algorithm graph will contain as low dependencies count as possible and will be more spread in width than in length.

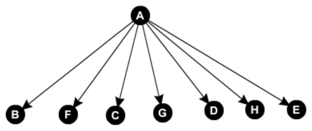


Fig. 2.8: Preferred computation graph

To evaluate the possible speed of algorithm assuming that every computational action takes 1 unit of the time,where T – time.

$$T(n) >= W(n) \tag{2.8}$$

If the computational graph contains a few dependencies and several vertices can be given to different processors, which by assumption run at one speed, then $T_p(n)$ can be expressed as

$$T_p(n) >= \frac{W(n)}{p} \tag{2.9}$$

where p is a count of the processors in a system. Taking into account the fact, that some vertices in a graph can have dependencies the longest dependency chain will affect the resulting time of computation or, in other words, Step complexity of the calculation will play very significant role. Overall computational time of the algorithm can be expressed as

$$T_p(n) = max[D(n), \frac{W(n)}{p}] \tag{2.10}$$

which is actually a demonstration of Brent's theorem [21]

Computer with amount of processors P can perform calculations with a time T_p less or equal than the number of processors needed to exploit the maximum concurrency in the algorithm.

Divide-and-Conquer

Divide-and-Conquer strategy is based on the splitting of the bigger problem into several sub-problems and executing sub-problems in the parallel. In ideal situation every sub-problem is an independent process with no need to communicate with other processes or all the communication is limited to the communication with the main (parent) process.

An example of a divide and conquer algorithm is Mergesort when a task is recursively broken down into sub-tasks, and sub-tasks results are combined together to produce the final result.

Typical way to express parallel Divide-and-Conquer pattern is given in the listing below:

```
ExpectedResult GetExpectedResut(InputData inputData) {
   if (inputData.Lenght < SEQUENTIAL_TRESHOLD)
      return GetExpecedResultSequentially(inputData);
   else
   {
     ExpectedResult _left, _right;
     PARALLEL INVOCATION
     {
       _left = GetExpectedResut(ExtractLeftHalf(inputData));
       _right = GetExpectedResut(ExtractRightHalf(inputData));
     }
     SYNCRONIZE;
     return Merge(_left, _right);
   }
}
```

Mergesort sorts inputData array. After parallel invocation of GetExpectdResults methods for the both parts of array Merge method is invoked. SYNCRONIZE block esures that Merge will take place only after inputData left and right sides are processed. It is obvious that the bottlneck of the MergeSort algorithm the is a Merge part. The Work and Step complexity of the Merge part is O(n). Thus, for the method, given in the code listing, the Work of the whole algorithm can be evaluated as

$$ER_1(n) = 2ER_1(\frac{n}{2}) + O(n) = O(nlogn) \tag{2.11}$$

Parallel Patterns

Parallel programming patterns were described in different sources, this work refers to the pattern described in Michael D. McCool paper [22]. In scope of this work the most interesting patterns are:

- Map
- Reduce
- Scan

Reduce. The reduce operation is an operation of the gathering of several values into one value. For instance the example of the reduce operation is a calculation of the sum of the all elements in array.

```
var sum = 0;
for(int i=0; i < inputData.Lehgth; i++)
{
   sum += inputData[i];
}
```

In this serial version of the code on each iteration the value of the variable *sum* is increased. There are several computational operation behind $+=$ operator: 1 – the current value of the *sum* variable is read out of the memory, 2 – *sum* value is summed together with the value of the $inputData[i]$ element, 3 - new value of *sum* is placed into memory. Overall the result of each new iteration depends from the result of the previous iteration.

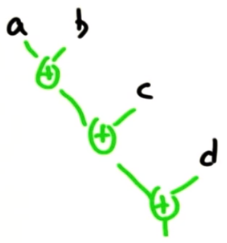


Fig. 2.9: Sequential Reduce

Having the list of elements $[A, B, C, D]$ it takes 3 calculations to make full reduce, which make it's Work complicity to be $O(n)$ performing $n-1$ operation. At the first look it is quite difficult to make such calculation in parallel because of dependencies between the iterations. Explicit shape of the calculation will look like that:

$$((A + B) + C) + D$$

Using operator associativity this equation can be rewritten as

$$(A + B) + (C + D)$$

and in that manner it is already possible to exploit parallel ways of reducing operation.

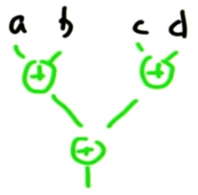


Fig. 2.10: Parallel Reduce

Both of given graphs have 3 units of Work, but the longest path through the graph, or Step complexity is reduced. As it is seen in the figure 2.10, potentially the parallel implementation should finish faster. Calculation of the needed steps count is done in the table 2.1

Tab. 2.1: Steps to n

n	Step Count
2	1
4	2
8	3

From the table 2.1 it is seen, that for the Parallel reduce the step complexity is $O(log_2n)$ which is very good speed up in productivity.

Scan. Another important parallel primitive is Parallel scan. Scan is an operation of pairwise calculation of the array element sums. As the result of each next calculations strongly depends on the result of the previous calculation, scan is quite difficult to implement in parallel. Given approach shows, how it is possible to exploit a parallelism in such series of algorithms. The scan chart is given in the Figure 2.11. Every member in the Out part of the array is calculated as a sum of the elements at positions i and $i + 1$

General flow of the Scan can be presented as pseudo code, which is given below:

```
var startingElement = identityElement;
var outputData = new []{};
for(int i=0; i < inputData.Lehgth; i++)
{
   startingElement = startingElement OPEATION inputData[i];
   outputData[i] = startingElement;
}
```

Scan operation uses an *identityElement* – element, which while being used with selected operator does not change the value of the second operation participant, for example in multiplication $identitElement = 1$, in logical AND $identityElment = true$, in logical OR $identityElment = false$. The serial implementation of Scan Work and Step complexity is $O(n)$. There are two types of scans: Inclusive scan – the result is calculated as: [reduce(a1), reduce (a1, a2), ..., reduce(a1, ..., an)] – output array will be the result of applying the reducing operation on all the previous elements including i–th element. Exclusive Scan - the result is calculated as: [reduce(a1), reduce (a1, a2), ..., reduce(a1, ..., an - 1)] – output will contain result of reduce operation of all the previous elements except the i–th element.

Scan operation, performed over array of elements, can be made in parallel using the pairwise reductions of respective array elements.

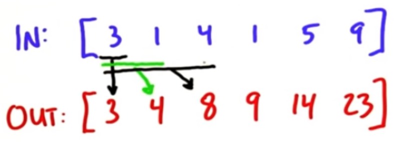


Fig. 2.11: Parallel Scan

Map. Map mean mapping of the one element to another. In Map pattern some function is applied to each element of the existing collection creating a new collection with the results of function calls. The order of function call over input array is not defined, which means it can run in parallel. If executable functions are pure and do not cause any side-effect (do not share states or have common variables), it is possible to talk about Step and Work complexity of $O(1)$. The real operations are bit more complicated, than simple mapping of the one array into another, thus, Map usually serves as an intermediate operation between other parallel primitives.

2.3 Distributed Computing

Calculations, done with participation of the several computation resources, connected via shared resources (memory, network, services) are identified as distributed calculations. Distributed systems include a lot of computational resources, which work together to give an appearance of single coherent system [23]. Some examples of Distributed systems are:

- Distributed Computing Systems
- Distributed Information Systems
- Distributed Pervasive Systems
- Multi-Agent Systems

2.3.1 Distributed Computing Systems

Cluster Computing Systems

Cluster computing is an easy way of doing high-performance computations by linking together relatively simple computers in high-speed network [24]. In almost all cases cluster computing is used to make parallel computations by running the same instance of one compute-intensive program at different machines simultaneously in data-parallel mode.

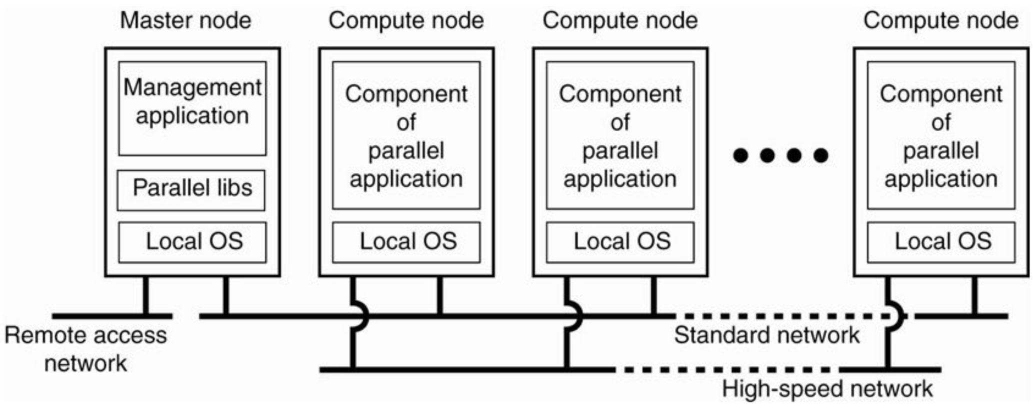


Fig. 2.12: An example of cluster system

Usually computational cluster consists of similar computers, with same operational system and capable of performing similar operations.

Grid Computing Systems

In Grid Computing systems computational resources can vary a lot (different operational systems, different file systems, different naming conventions) and might need additional effort for consolidation [25]. Grid computing uses layered approach for the systems collaboration. Each set of resources is grouped into a layer and these layers are responsible for particular set of calculations.

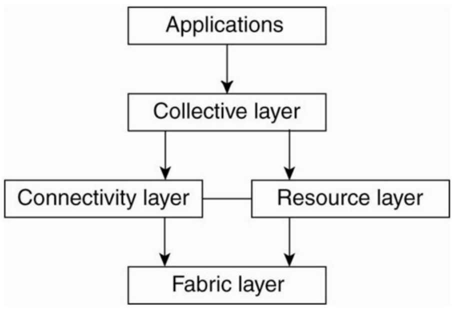


Fig. 2.13: Grid Computing systems layers

Every layer is responsible for handling of different resource types:

- Fabric Layer – provides interfaces to the local systems in specific environment
- Connectivity Layer – provides the set of protocols, needed for communication between different resources.
- Resource Layer – responsible for a single resource. Uses the functions provided by the connectivity layer and calls the interfaces made available by the fabric layer
- Collective layer – handles access to multiple resources
- Application layer – layer, which makes Grid Computing system to be available for the end-user

2.3.2 Multi-Agent Systems

As a one example of the Distributed computing, the idea of the distributed computations using Multi-Agent systems can be taken in use. In the prof. Taveter's book "The Art of Agent-Oriented Modellng" Multi Agent system

[26] is defined as a set of Agent entities, which are grouped together to make a complex entity, capable of performing some complicated actions. Agent is defined as an entity that performs a specific activity in an environment of which it is aware of and that can respond to changes.

An agent is a program, which acts in accordance with some predefined protocol, is capable of making decisions on the basis of the existing knowledge base and is able to gather and share behavioural data. Thy typical software agent architecture can be found in the book “The Art of Agent-Oriented Modeling” [27].

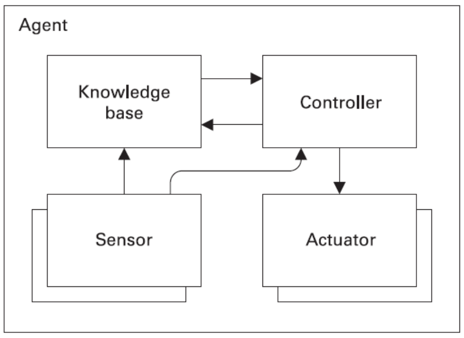


Fig. 2.14: An abstract agent architecture

Currently, many research labs, universities, companies and industrial organizations are working in this field, and the list of those companies are constantly expanding. There are a few well-known names and small groups, already recognized research labs and organizations (such as the Carnegie Mellon University [28]), as well as huge multinational companies such as Apple [29], Daimler AG [30], HP [31], IBM [32], Microsoft [33], Oracle [34]. In practice agent-based technologies are used in information management such as workflow management and network management, air traffic control, information retrieval, e-commerce, banking, stock operations, training and education, electronic libraries and many other fields. In this work agents will be applied to commit scientific modelling operations using the large set of input data.

Agent entities within multi-agent system can act as computational units for solving of large computational task. Agent has an ability to learn and share information with other agents. It is quite simple to organize a lot of agents located over some network into one computation cluster and put them to calculate chunks of data. Figure 2.15 shows the example of multi-agent system.

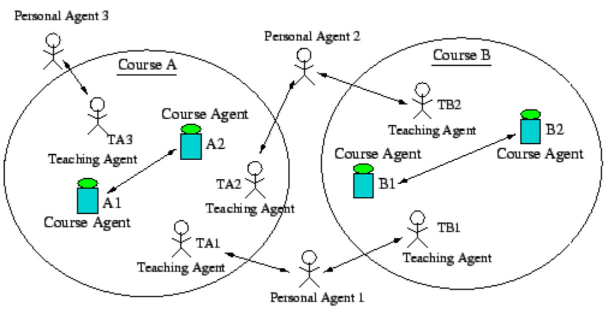


Fig. 2.15: Example of Multi Agent System

2.3.3 Crowd Computing

In the recent years Crowdsourcing has been one of the buzzwords. In most cases crowdsourcing is defined as the outsourcing of tasks to a large group of people instead of assigning such tasks to an in – house employee or contractor [35]. Technically some tasks are given away to the groups of the people, who do the work either for free or for a very small fee.

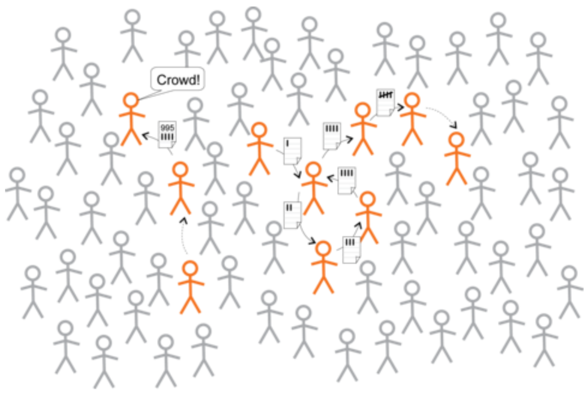


Fig. 2.16: Crowd computing

One of the first cases of crowdsourcing can be considered a compilation of the Oxford dictionary – editors used volunteers to participate in finding all the words in a language and determining their values. Crowd computing is a branch of the crowdsourcing, which takes advantages of remote computational resources on the basis of benevolence. In practice crowd computing means the distribution of the payload between parties, who have given a permission to use their computational resources to perform some tasks.

Crowd Computing works with an external crowd (public crowd computing) and internal (private crowd computing). The principles of work remain the same, the main difference is in the tasks, which are solved by different crowds. Although Crowd Computing is quite young and new, there are already many success stories of utilization and some successfull attempts of commercial use – by the end of the year 2014 about 30 companies were offering commercial products and services to its customers using Crowd Computing. This companies work in different sub-segments of Crowd Computing. For example, Workfusion [36] from the United States and Hong Kong Cloud Factory [37] work with sub-segment of Crowd Management Application.

Another example in the utilization of Crowd Computing – the Sony Playstations were used to build a computational cluster. [38]. That cluster does not belong to the TOP 500 performance computing systems, but still has a remarkable power. Another scientific experiment – Seti@Home uses internet connected computers in the Search for Extraterrestrial Intelligence (SETI). Everyone can participate by running a free program that downloads and analyzes radio telescope data. [39]

Current work applies Crowd Computing principles (described in Section 2.3.1). Software units for option pricing will be created and suggested for usage in several available machines in corporative network utilizing the idea of private crowd computing. Software units will be sent to specified computers with the help of a special trigger, capable of discovering and utilizing shared resources in a network. For the convenience, the parts of the system which are sent out to other computers are called the *agents*, although they do not have all known intelligent agent features.

2.3.4 Other types of Distributed Systems

There are a lot of distributed computing systems, which do not fit into the context of the current work. For example Transaction procressing systems. Such systems are used as facilitators between distributed databases and software applications [40].

3. RELATED WORK

This chapter describes the state of art in the field of the financial derivatives pricing field, gives brief overview of the financial derivatives valuation with respect to underlying assets price using numerical methods and pricing models. Monte-Carlo approach is mentioned. Some systems for model-based price forecasting are described. The chapter contains a description of existing generic parallel and distributed computing frameworks and systems. Cluster systems, MapReduce pattern and Crowd computing are brought out as a possible solution to the stated problem.

3.1 Derivatives Pricing

3.1.1 Numerical Methods

Along with existing Derivatives pricing methods, such as Black-Scholes [2], binomial method [3], analytical and investment companies widely use in-house pricing models of possible price changes. These models reflect behaviour of specific assets with multidimensional relations. For example Norwegian company Rystad Energy [41] uses simulation modeling to make forecasting of Crude Oil price for High, Mid, Low and Low Low price scenarios.

Derivatives pricing is based on the need to constantly follow the asset prices. That means there is a need to simulate as much price fluctuations as it is possible. For that purpose numerical methods and Monte-Carlo techniques are used. Monte-Carlo simulations provide statistical overview of all possible price changes and the method itself can be easily implemented in parallel way.

3.1.2 Pricing Systems

With a development of parallel and distributed computing techniques and computational hardware, some platforms oriented to the financial derivatives pricing problem were implemented. Schryver, Shcherbakov, Kienle and Wehn proposed a system based on the FPGA and Heston model [42]. The work covered creation of the hardware prototype with respective parallel implementation of pricing model. The prototype of the system in comparison to simulation with Intel Core i5-3320M CPU performed 38 times faster.

Another sample is PicsouGrid [43], a framework for Grid computing, written in Java. Was present in 2007 at International Simposium of Grid computing. It follows the Data Parallel approach with it's server and workers abstraction. Contains Simulator interface and some implementations. In case of need own implementation can be defined.

A lot of companies have their own success stories of the Derivatives pricing using in-house models and existing generic parallel and distributed computing solutions.

3.2 Parallel and Distributed Solutions

Systems, which do simulation modeling on a basis of big amount of data and expensive computations are used in a wide variety of domains. The most interesting systems in scope of this work are the systems, which allow to set up the custom model of some process and distribute it over the numerous computational resources in a fast, cheap and reliable way. The set of most well-known systems, which can perform tasks, similar to the one, addressed in this work are given below:

- Apache Hadoop
- Dryad
- Nokia Disco
- BOINC

3.2.1 Hadoop

The Apache Hadoop is a framework that enables the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from a single server to thousands of machines, each offering local computations and storage. Apache Hadoop is a set of products:

- Hadoop Distributed File System
- Hadoop YARN – framework for job scheduling and cluster resource management
- Hadoop MapReduce – YARN-based system for parallel processing of large data sets
- Hadoop Common – package of common tools for managing other modules

The main component and a trade mark is Hadoop Distributed File System, which, like regular file systems, consist of data blocks and descriptor tables with the difference that in Hadoop Distributed File System the role of descriptor tables fulfils NameNode and files data is spread across DataNodes. For each file NameNode contains a name, blocks and paths to blocks. To increase reliability file blocks are replicated.

Apache Hadoop distributes computation across the workers, each worker fulfils given part of the computations locally. The main focus of Hadoop is to work with a big unstructured and constantly growing data files. Under the hood Apache Hadoop is based on the MapReduce paradigm (see Secton 3.2.5).

3.2.2 Microsoft Dryad

Dryad is a distributed execution engine that simplifies the process of implementing data-parallel applications to run on a cluster. Dryad has been deployed by Microsoft since 2006 and is used daily to analyze petabytes of data on Microsoft codename *Cosmos* clusters consisting of thousands of computers. The existing public release of Dryad runs on Windows HPC clusters. The original motivation for Dryad was to execute data mining operations but later it became a general-purpose execution engine which could be used to implement a wide range of other tasks, including time series analysis, image processing, and a variety of scientific computations [44].

Dryad approach is to map a distributed system as a directed acyclic graph, where each vertex would present some computational primitive – a *job*. Graph would be used in computational environment to put together a cluster system. Each vertex would run at least one computation job. A job content could vary from specially designed programs to legacy executables, which should be possible to incorporate into Dryad system. Figure 3.1 shows the Dryad basic distributed job.

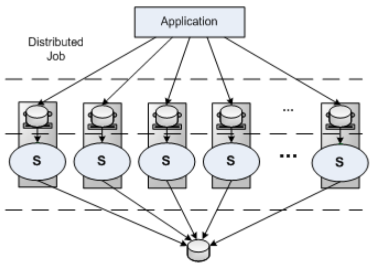


Fig. 3.1: Dryad distributed job

As Apache Hadoop, Dryad is also a follower of the MapReduce Pattern. Dryad project was closed by Microsoft due to competing with Hadoop and Microsoft decided to focus on Azure and Windows Server-based version of Apache Hadoop [45].

3.2.3 Nokia Disco

Another similar framework for distributed computing is Nokia Disco. Disco is an implementation of MapReduce for distributed computing. [46].

3.2.4 BOINC

A software platform, which is designed to support scientific project, in scope of which radio telescope data is downloaded and analysed [47]. The project SETI@HOME [39] connected computers on voluntary basis and every computer performed some part of needed calculations. BOINC system kept track, how much CPU time each joined node contributes to the platform. BOINC is known as MapReduce in Volunteer environment [49].

3.2.5 MapReduce

MapReduce [50] is a common programming model for the systems described above. MapReduce is a model of distributed computations, presented by Google. It is used in processing of large data sets in computational clusters. In general, the work of MapReduce contains of two steps – Map and Reduce. Map does the processing of the input data. In classical implementation of MapReduce one part of the system always serves as a master node which serves the subtasks to worker nodes. Reduce step involves "deflating" of the calculation results, received by the master node from worker nodes. The typical MapReduce workflow is shown in the Figure 3.2

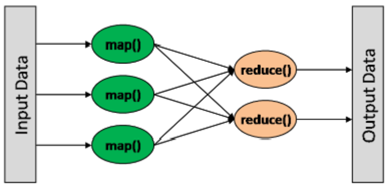


Fig. 3.2: Map Reduce

Google issued a framework with the same name. Framework is based on the Map and Reduce functions, which are widely used in functional programming [50].

To reach the goals of this work, the insights of all these systems are combined together:

- MapReduce programming model is selected as a main approach to solve the stated problems.
- Map and Reduce primitives are empowered by the application of Parallel Computing – Parallel Map and Parallel Reduce.

- To create a cluster from available computers, a method proposed by Dryad research project team is used.
- Voluntary calculations approach used with BOINC is used for creating of private calculation crowd.
- The principles of calculation clustering with MapReduce are examined for the applicability with relational database models.

4. SYSTEM DESIGN AND IMPLEMENTATION

This chapter contains the proposal of solution to the problem stated in Section 1.3. Chapter describes the high-level design of the distributed system, which utilizes MapReduce pattern and the principles of Crowd Sourcing. The implementation section contains a description of computer cluster creation on the basis of available computation resources. Each available resource runs asset price forecasting program employing Monte-Carlo approach. At the phase of prototyping initial tests are done to determine the best program parallelism options.

Starting point for the prototype – local network of small analytical company, consist of 40 personal computers and 4 servers. Company uses in-house mathematical models and custom application to create Option price forecast. Forecasting application is a hand-made application, which runs on one of the company servers nightly. Every night is takes at least 7 hours to calculate a forecast, which is later placed into business intelligence data storage.

The idea is to exploit the data-parallel pattern for increasing the productivity of existing forecast application by introducing calculations on different machines – Cluster Computing System. Cluster will consist of computational agents, which are sent to all servers and to the personal computers of the company employees (Private Computing Crowd). Instead of running the application on one relatively powerful server, all available computing resources are participating in calculations. The idea of the system proposed in this work is in utilization of the methods, described previously, for creation of a cluster on the basis of the available computers, allowed to be involved into calculations by their owners.

4.1 System Summary

To achieve the goals, established at the beginning of this work it was decided to design a prototype of a dynamic computation system basing on ideas of Crowd Computing 2.3.3 and Cluster Computing Systems 2.3.1. System will calculate Option underlying assets price forecast using one of methods described in Section 2.1. During Option price calculation the time series of underlying asset price will be used. The lowest granularity level is 1 day.

Figure 4.1 shows the schematic outline of price forecast calculation. The "Asset Input" data set holds all known information about the option underlying assets. All asset updates are made to the Asset Input data set. Assumptions (prices, dates etc.) and model parameters (cost elements and levels, country factors, technology factors, etc.) are fed into tables and used in processing. Processing is done asset by asset, after that the "Asset Output" data set is populated with calculated price forecast data for each of existing assets.

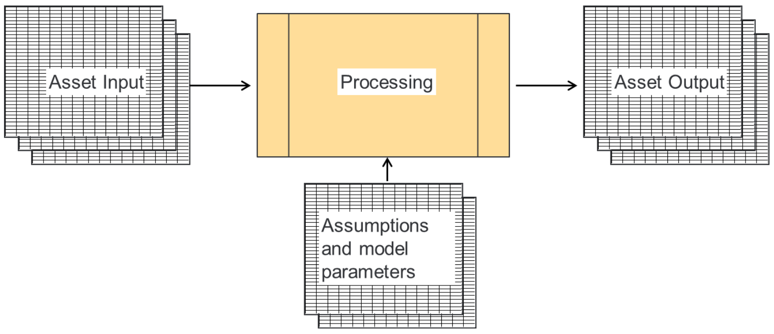


Fig. 4.1: Schematic description of Forecasting System

The system uses the data available for free at Yahoo Finance [51] and Option price calculation will be done using the model described in Section 4.3.

4.2 Asset Data

Asset data is updated every day, which implies the need to recalculate underlying asset price forecast daily taking into account the dynamics of pricing at the lowest granularity level. Table 4.1 shows the Asset data structure.

Tab. 4.1: Asset Data

Asset Details	**Comment**
Name	Asset Name
Date	Historical date
Open	Opening trading price at the Date
High	Highest trading price at the Date
Low	Lowest trading price at the Date
Close	Closing price at the Date

4.3 Pricing Model

Current work assumes that data from opened sources, is good enough to calculate Option underlying assets price forecast using MonteCarlo approach [10] and mathematical model partly based on the Black-Sholes [2] equation.

Monte-Carlo simulation can be easily wrapped into a scalable distributed application, which encapsulates parallel solution for the model calculations. Although, as mentioned above, the calculation of the price of the European option is very convenient to carry out using the analytical method, Monte-Carlo method will be used as the most suitable method to demonstrate achievement of the objectives of this work.

Modelling purpose is to forecast the price of Option at some moment t Model operates on the basis of historical prices data $P = [P_0, P_1.....P_t]$ and given strike price X and

1. Daily return rate can is calculated as

$$dr = \ln(\frac{P_t}{P_{t-1}}) \tag{4.1}$$

2. Average of daily return rates

$$\mu = \frac{1}{n}\sum_{i=1}^{n} dr_i \tag{4.2}$$

3. Variance of the daily return rates

$$var = \frac{\sum(x - x_{avg})^2}{n} \tag{4.3}$$

4. Standard deviation of daily return rates

$$stddev = \sqrt{\frac{\sum(x - x_{avg})^2}{n}} \tag{4.4}$$

5. Asset drift

$$drift = \mu - (\frac{var}{2}) \tag{4.5}$$

6. Asset future price calculation

$$P_t = P_{t-1}\, e^{(drit+var*B_t)} \tag{4.6}$$

7. Option price for 1 iteration is calculated using equation

$$C_t = e^{-r(t)} E\, max[(0, P_t - X)] \tag{4.7}$$

8. Option price estimation is done on the basis of the mean value of calculated price series using

$$C_{mean} = \frac{1}{n}\sum_{i=1}^{N} C_i(P,T) \tag{4.8}$$

This model is implemented as a set of kernels, responsible for calculating of the needed features (Both in serial and parallel manner).

4.4 System Initial Requirements

System high-level description is given as following: as a financial analyst I would like to have a system for Option price forecasting on the basis of the underlying assets historical data. System has to be easy to run and maintain. System should provide a result within appropriate time.

Basing on that description it is possible to define the requirements more precisely. There are four main possible aspects client might look for:

- Performance – system has to deliver result within a specified time limit
- Reliability – downtime per specified timespan amount should not exceed specified number

- Scalability – system has to be easily extendable
- Maintainability – system should be easily maintainable

Below, only top-level requirements were listed. Schematic requirements model is presented in the Figure 4.2.

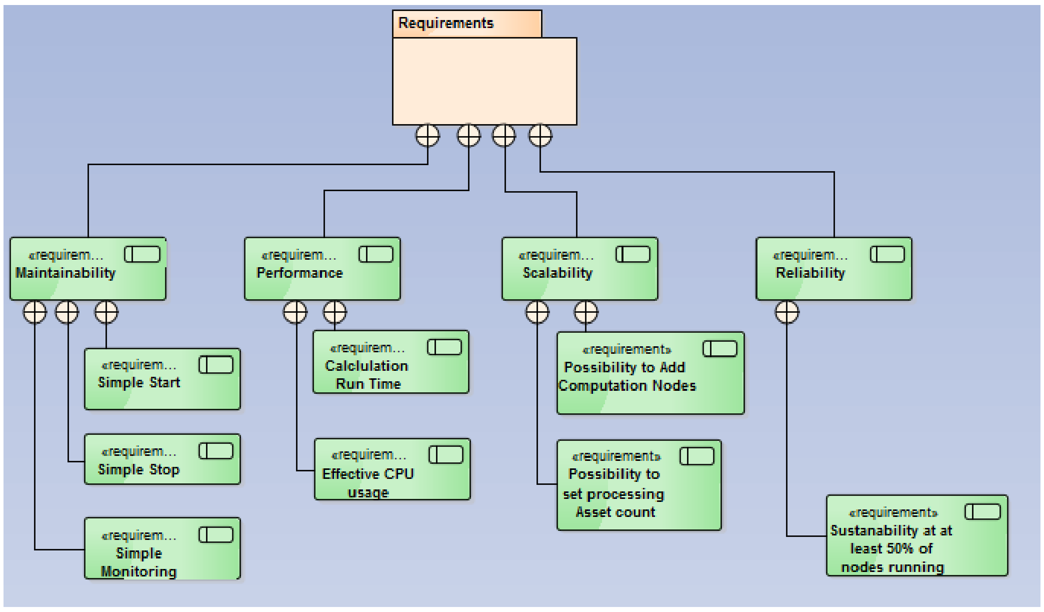

Requirements
«requirem... Maintainability
«requirem... Performance
«requirem... Scalability
«requirem... Reliability
«requirem... Simple Start
«requirem... Simple Stop
«requirem... Simple Monitoring
«requirem... Calclulation Run Time
«requirem... Effective CPU usage
«requirement» Possibility to Add Computation Nodes
«requirement» Possibility to set processing Asset count
«requirement» Sustanability at at least 50% of nodes running

Fig. 4.2: Requirements model

4.5 *High-level Architectural Overview*

The task of calculation of the Option prices on the basis of historical data is a task of producing new data rows from the old data rows using mathematical model. Mathematical model does a specified action on every historical data row, which correspond to Map pattern. Then a large set of historical data is used for creation of one or more rows of future prices, this corresponds to Reduce pattern. Suitable algorithms with the justification of the choice are given in Section 2.2.4. The processing of the historical rows using Monte-Carlo simulation will be done in parallel on a single node

Every node of the system, implementing MapReduce pattern is able to run independently on every available computer in the local area network, leading to additional parallelism and performance gaining by distributing the calculation units across available machines, creating computing cluster, described in Section 2.3.1.

An account the system is running, should be granted with appropriate rights to access data storages and computing resources. High-level presentation of the system architecture is given in the Figure 4.3

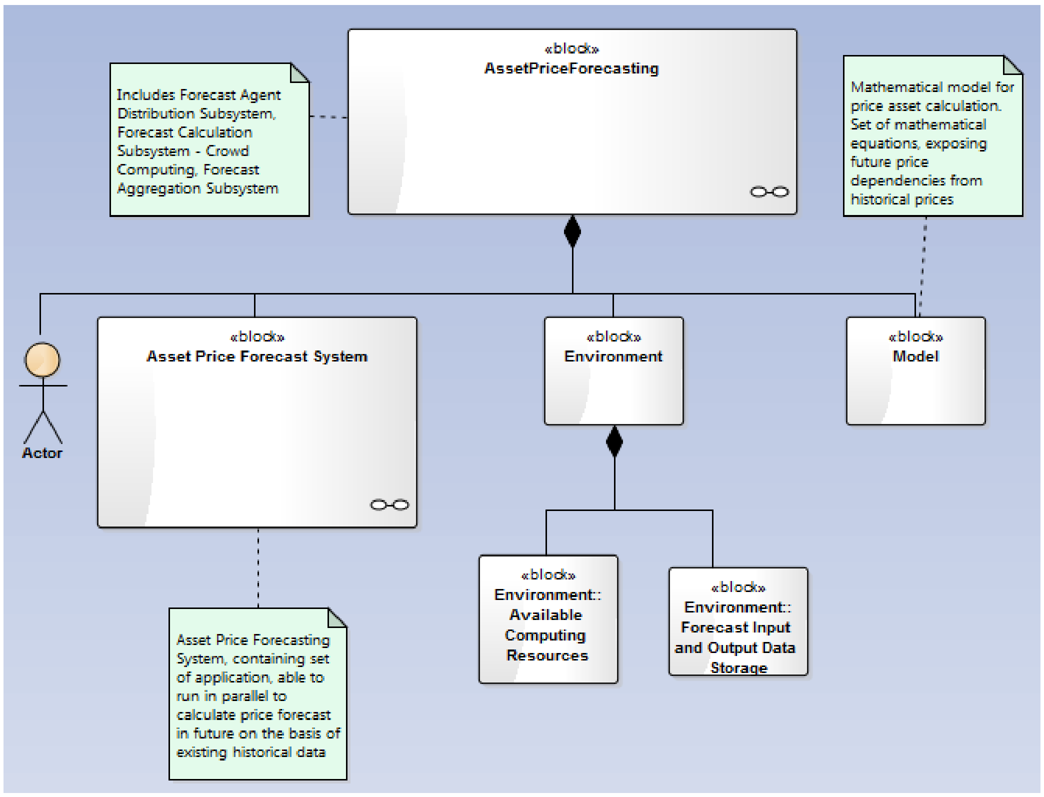

«block»
AssetPriceForecasting
Includes Forecast Agent Distribution Subsystem, Forecast Calculation Subsystem - Crowd Computing, Forecast Aggregation Subsystem
Mathematical model for price asset calculation. Set of mathematical equations, exposing future price dependencies from historical prices
Actor
«block»
Asset Price Forecast System
«block»
Environment
«block»
Model
«block»
Environment:: Available Computing Resources
«block»
Environment:: Forecast Input and Output Data Storage
Asset Price Forecasting System, containing set of application, able to run in parallel to calculate price forecast in future on the basis of existing historical data

Fig. 4.3: High-level architecture

Classical cluster consists of fairly the same type of machines, and usually the amount of these machines is known, however each company usually has a lot of personal workstations, which run during working hours and stay idle more than 10 hours per day. This work proposes to take in use those machines on the basis of benevolence – to compose Private Crowd, described in Section 2.3.3. Literally, each computer in the local area network can be considered as computational resource and it's owner can accept or decline the computational task. For those who will accept a task, software agent can be added remotely.

System idea is applicable for every software – and database server platform. In scope of that work the relational database is used. It is not strictly required, but should satisfy the need for data structuring need and data delivery options from the storage to calculation part.

4.6 System Design

For demonstration of the selected approach, Cluster computing system prototype is designed and evaluated for the performance, usability and development costs.

For the Price Forecasting System main four subsystems are required:

- Forecast Agent Sending System – refers to Scalability requirement in the Figure 4.2. The goal of the system is to send a calculation agent to specified computer in the computation environment.
- Forecast Triggering Subsystem – refers to the Maintainability requirement in the Figure 4.2. Subsystem provides the interface for start, stop and monitoring of the whole system.

- Forecast Calculation Subsystem – refers to the Performance and Reliability requirements in the Figure 4.2. Subsystem is responsible for Price Forecast delivery within a specified time.
- Forecast Aggregation Subsystem – integration with data visualization or analysis systems. Out of the scope of this work and not covered in a full details.

Schematic representation of the system is given in the Figure 4.4.

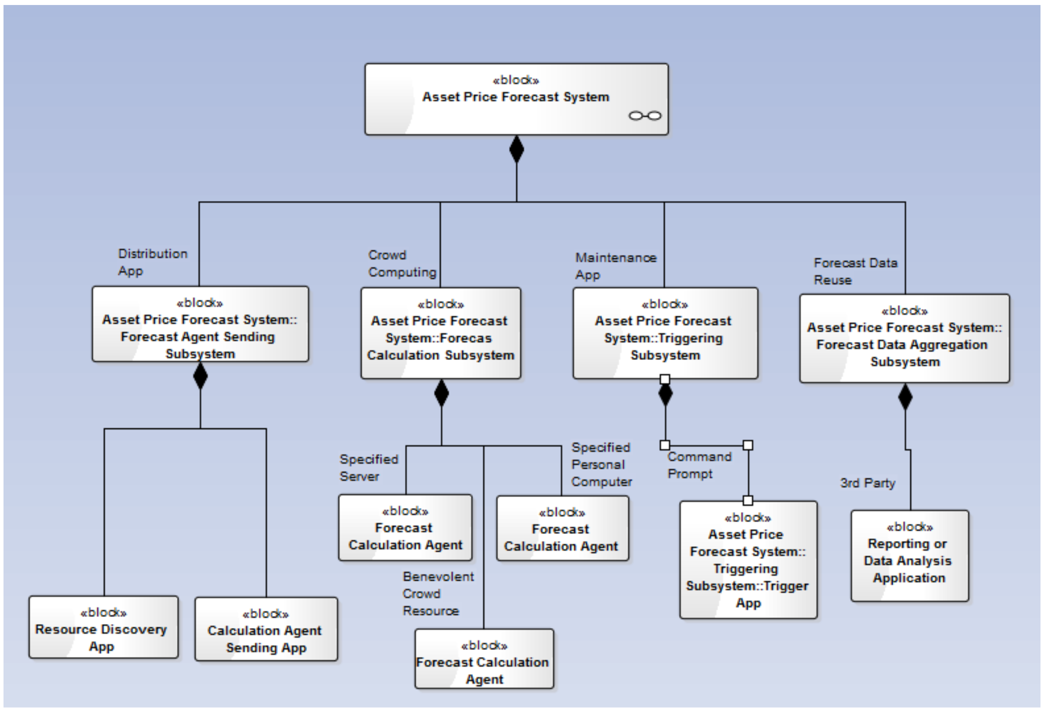

«block» Asset Price Forecast System
Distribution App
Crowd Computing
Maintenance App
Forecast Data Reuse
«block» Asset Price Forecast System:: Forecast Agent Sending Subsystem
«block» Asset Price Forecast System::Forecas Calculation Subsystem
«block» Asset Price Forecast System::Triggering Subsystem
«block» Asset Price Forecast System:: Forecast Data Aggregation Subsystem
Specified Server
Specified Personal Computer
Command Prompt
3rd Party
«block» Forecast Calculation Agent
«block» Forecast Calculation Agent
«block» Asset Price Forecast System:: Triggering Subsystem::Trigger App
«block» Reporting or Data Analysis Application
Benevolent Crowd Resource
«block» Resource Discovery App
«block» Calculation Agent Sending App
«block» Forecast Calculation Agent

Fig. 4.4: System Design

Forecast Agent Sending Subsystem

Although the idea of Cluster Computing Systems is not new, classical cluster computing system can be empowered using Crowd Computing [35] principles. Some users willing to help with calculations can join their computers into the cluster. The Forecast Agent Subsystem goal is to discover and deliver such computers into the computation environment. Figure 4.5 shows the use case model for Agent Sending Subsystem.

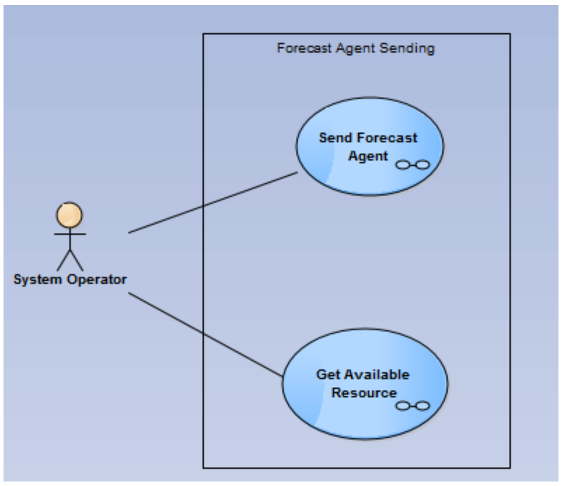


Fig. 4.5: Forecast Agent Sending Subsystem

Primary use cases of the system are given below.

Tab. 4.2: Use case: Send Forecast Agent

Use Case	Send Forecast Agent
Primary Actor	System Operator
Scope	Computational Environment
Brief description	Forecast application sending to available computational resource – server or personal computer
Preconditions	Server name and network path are known
Postconditions	Forecast calculation application is placed on a specified computer.

Tab. 4.3: Use case: Get Available Resource

Use Case	Discover Available Resource
Primary Actor	System Operator
Scope	Computational Environment
Brief description	Get information about available computers with shared resources in computational environment
Preconditions	Primary Actor has appropriate access rights
Postconditions	List of resources returned

Forecast Calculation Subsystem

The core of the system. Price Forecast Calculation application, which utilizes data-parallel software development pattern described in Section 2.2.4. Application utilises the best practises of Parallel programming described in Section 2.2.4 and, in particular, parallel Map and Reduce primitives. The application is distributed across computational environment and works simultaneously using cluster computing approach. Cluster system utilizes Crow computing approach.

Use cases of the Forecast Calculation Subsystem are given in Figure 4.6

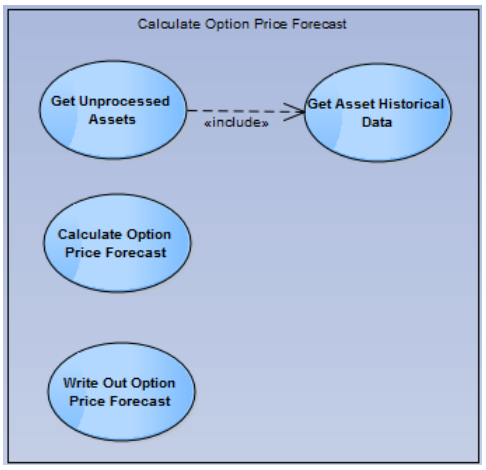

Calculate Option Price Forecast
Get Unprocessed Assets
«include»
Get Asset Historical Data
Calculate Option Price Forecast
Write Out Option Price Forecast

Fig. 4.6: Forecast Calculation Subsystem

Use cases definitions:

Tab. 4.4: Use case: Get Unprocessed Assets

Use Case	Get Unprocessed Assets
Primary Actor	Forecast App
Scope	Computational Environment
Brief description	Forecast Application gets the Assets available for calculation from the database and marks them as participating in calculation
Preconditions	not calculated Assets present in the data storage
Postconditions	List of Assets transferred for calculation, in data storage an asset marked as participating in calculation

Tab. 4.5: Use case: Get Asset Historical Data

Use Case	Get Asset Historical Data
Primary Actor	Forecast App
Scope	Computational Environment
Brief description	Forecast calculation application gets the price time series for the asset, which participates in the Option price forecast calculation
Preconditions	Price time series for selected asset are available in the data storage
Postconditions	Price time series for selected asset are delivered to Forecast calculation application

Tab. 4.6: Use case: Calculate Option Price Forecast

Use Case	Calculate Option Price Forecast
Primary Actor	Forecast App
Scope	Computational Environment
Brief description	Forecast application performs multiple calculations of underlying assets price using Monte-Carlo method and chosen mathematical model
Preconditions	Price time series for selected asset are available in the data storage
Postconditions	New price time series are generated in forecast application

Tab. 4.7: Use case: Write Out Option Price Forecast

Use Case	Write Out Price Forecast
Primary Actor	Forecast App
Scope	Computational Environment
Brief description	Option Price Forecast and new generated asset time series writing to data storage
Preconditions	Selected asset is calculated
Postconditions	New price time series are inserted to the data storage

Forecast Triggering Subsystem

Forecast Triggering Subsystem, along with the Forecast Agent Sending Subsystem defines an interface for operating and maintenance of the Forecasting System. The main goal of this system is to start, stop and monitor price forecasting system in a flexible way – on all available cluster nodes together or on one particular node. It also has to capture the current state of the whole entire forecasting system or one particular running Forecasting Agent on some specified computation node of the cluster.

The use case model of this subsystem is given in Figure 4.7

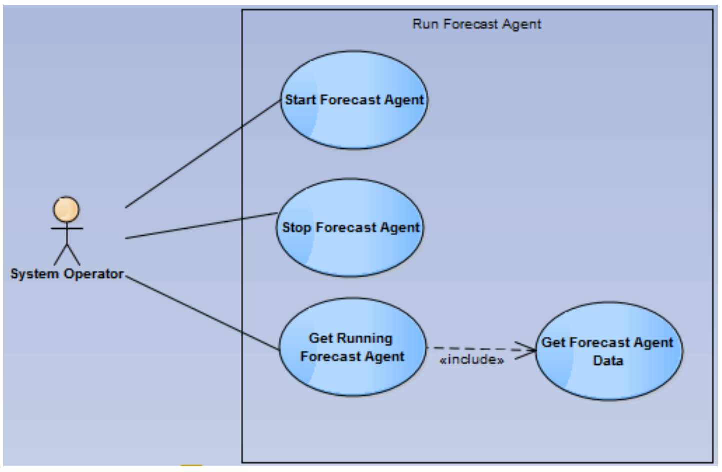


Fig. 4.7: Forecast Triggering Subsystem

Forecast Triggering Subsystem's Use Cases:

Tab. 4.8: Use case: Start Forecast Agent

Use Case	Start Forecast Agent
Primary Actor	System operator
Scope	Computational Environment
Brief description	Start Forecast Computation
Preconditions	Forecast Agent is available at selected computing resource
Postconditions	Forecast Agent is running at selected computing resource

Tab. 4.9: Use case: Stop Forecast Agent

Use Case	Stop Forecast Agent
Primary Actor	System operator
Scope	Computational Environment
Brief description	Stop Forecast Computation
Preconditions	Forecast Agent is running at selected computing resource
Postconditions	Forecast Agent is stopped at selected computing resource

Tab. 4.10: Use case: Get Running Forecast Agent

Use Case	Get Running Forecast Agent
Primary Actor	System operator
Scope	Computational Environment
Brief description	Connect to running Forecast Agent and read current state
Preconditions	Forecast Agent is running at selected computing resource
Postconditions	Forecast Agent current state is read out

Tab. 4.11: Use case: Get Forecast Agent Data

Use Case	Get Forecast Agent Data
Primary Actor	System operator
Scope	Computational Environment
Brief description	Grant connectivity to the running Forecast Agent in order to read current state
Preconditions	Forecast Agent is running at selected computing resource
Postconditions	Forecast Agent current state is read out

Forecast Data Aggregation Subsystem

The last system in the design is a system for data aggregation. In general it includes the Option Price Forecast output data available in the data storage plus a standard interface for connecting to this data storage. It may be the connection to some data visualization or analysis applications. The main goal of this subsystem is to present the data in the shape needed for the users of the system, consequently, a use case for that part of the system can be defined as following:

Tab. 4.12: Use case: Get Option Price Forecast

Use Case	Get Option Price Forecast
Primary Actor	System User
Scope	Computational Environment
Brief description	Get calculated Option prices and new asset price time series
Preconditions	Forecast System is finished at all the calculations
Postconditions	Forecast data is sent to presentation layer

4.7 Implementation

The prototype presented in this Section is implemented in accordance to the system design and system includes all parts, described in Section 4.6

There are several available computational resources connected to the local area network. Along with these defined available resources there are other personal computers and laptops, used by the company employees and not fully loaded 24 hours per day. Specifications of available resources are given below.

Tab. 4.13: Computing resources

Server 1	
Processor	Intel® Core^{TM} i7-5930K CPU @ 3.5 GHZ
CPU Cores	6
CPU Threads	12
CPU Number	1
Server 2	
Processor	Intel® Core^{TM} i7-960 CPU @ 3.2 GHZ
CPU Cores	4
CPU Threads	8
CPU Number	1
Server 3	
Processor	Intel® Xeon® CPU X5660 CPU @ 2.6 GHZ
CPU Cores	6
CPU Threads	12
CPU Number	1
Server 4	
Processor	Intel® Xeon® CPU E5-2650 v3 @ 2.30 GH
CPU Cores	10
CPU Threads	20
CPU Number	2
Workstation	
Processor	Intel® Core^{TM} i7-4810MQ Processor
CPU Cores	4
CPU Threads	8
CPU Number	1

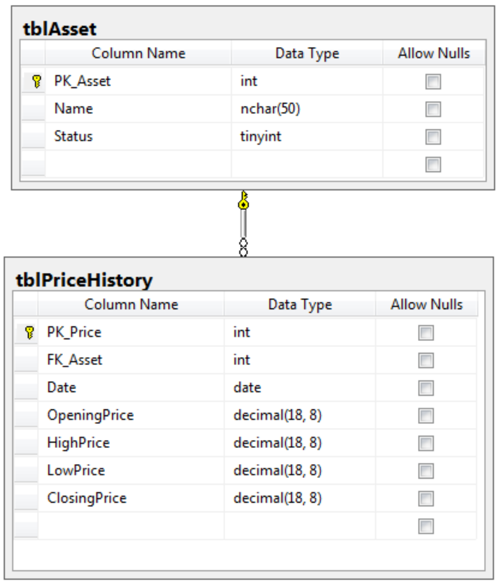
The cluster under consideration contains at least four described servers, which are always available, additionally it will use available workstations mentioned in table 4.13

The software is implemented using Microsoft.NET platform [52], which fits perfectly with the existing operating systems and provides a wide set of means for fast software development. In this work no special 3rd party high-performance computing libraries are used. Triggering system is responsible for the cluster managing and forecasting applications will perform Monte-Carlo simulations.

4.7.1 Asset Historical Data

MS SQL server 2012 SP1 version 10.0.300 [53] is used as a data storage for the prototype relational database.

Option underlying assets' historical data structure is described in Section 4.2. A relational database table structure is created in accordance to this definition. Asset data structure is split into two parts – common information, which holds Asset name, an internal id, a processing status and pricing time series of each asset. These parts are put into respective tables with assigning of the primary and foreign keys and relations between the tables. Split of an asset data structure is shown in Figure 4.8



tblAsset

Column Name	Data Type	Allow Nulls
PK_Asset	int	☐
Name	nchar(50)	☐
Status	tinyint	☐
		☐

tblPriceHistory

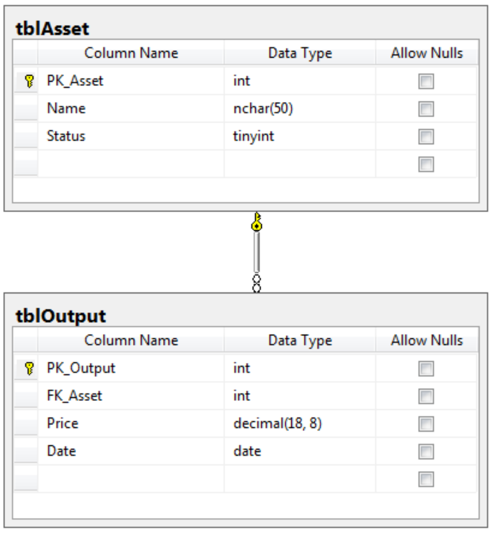
Column Name	Data Type	Allow Nulls
PK_Price	int	☐
FK_Asset	int	☐
Date	date	☐
OpeningPrice	decimal(18, 8)	☐
HighPrice	decimal(18, 8)	☐
LowPrice	decimal(18, 8)	☐
ClosingPrice	decimal(18, 8)	☐
		☐

Fig. 4.8: Asset Data Structure

This database diagram shows the initial data storage. Schematically there are two tables, sufficient to hold minimal set of data for the price forecast modelling. Relation between tblAsset and tblPriceHistroy is One-to-Many

- tblAsset – where `PK_Asset` is clustered index, Name is Asset name and Status is a system field, needed for signalling during the simulation
- tblPriceHistory – where `PK_Price` is clustered index, `FK_Asset` is a Foreign key to tblAsset and Opening, High, Low and Closing prices are respective prices for asset at some trading date in future.

Each asset is calculated against the mathematical model, calculating option price forecast for the amount of given days ahead. By assumption the number of the days in future is less than the number of historical price data entries, thus the output data of each asset contains reduced amount of entries (the implementation of parallel Reduce primitive). Output data table structure is almost the same as the tblPriceHistory structure. Linking between common asset data table and output table is the same as the linking between tblAsset and tblPriceHistory One-to-Many. Asset output data structure is given in the Figure 4.9



tblAsset

	Column Name	Data Type	Allow Nulls
🔑	PK_Asset	int	☐
	Name	nchar(50)	☐
	Status	tinyint	☐
			☐

tblOutput

	Column Name	Data Type	Allow Nulls
🔑	PK_Output	int	☐
	FK_Asset	int	☐
	Price	decimal(18, 8)	☐
	Date	date	☐
			☐

Fig. 4.9: Asset Output Data Structure

Extraction of the data from the database is done using stored procedures. Despite the fact of existing of the ORM frameworks for Microsoft SQL server (NHibernate, Entity Framework), SQL stored procedures are still the best fit to write applications, working with relational databases. There are a lot of advantages of stored procedures in comparison with plain SQL or ORMs, which literally generate plain SQL, such as easy maintenance, unified access, enhanced security, but the main one is, of course, performance – stored procedures are pre-compiled software units, which perform faster in comparison to SQL queries.

For prototype needs two main stored procedures are created:

- appGetAsset
- appGetAssetPrice

Stored procedure appGetAsset is a procedure, which receives an integer parameter, which means the number of the assets to extract from the database. This stored procedure returns a list of integers, representing the column `PK_Asset` with offset defined by the input parameter.

```sql
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN TRANSACTION;

SELECT top(@AssetCount) PK_Asset INTO #assets FROM [dbo].[tblAsset]
    WITH (ROWLOCK, XLOCK) WHERE [Status] = 0 order by PK_Asset;

UPDATE [dbo].[tblAsset] SET Status = 1 WHERE PK_Asset IN (SELECT
    PK_Asset FROM #assets)

SELECT PK_Asset from #assets;
DROP TABLE #assets;

COMMIT TRANSACTION;
```

In that query an asset number specified by input parameter is extracted. After that assets are marked into Processing state. As this procedure works with many applications simultaneously, it is implemented using locking mechanism, which ensures that only one instance of Forecasting Agent fetches the data from the table in order not to let same assets to leak to another instances of Forecasting Agent

Stored procedure appGetAssetPrice receives primary key of tblAsset as input parameter and fetches data from tblPricceHistory. Assuming the fact, that the data in that table is always static, no updates or deletes are planned during the simulation, to grant high level of readability select query is done with NOLOCK instruction, which tells the server not to lock the table during the execution of the asset price data selection by a Forecast Agent instance.

```sql
SELECT * from [dbo].[tblPriceHistory] WITH (NOLOCK)
    where FK_Asset = @PK_Asset order by Date asc;
```

Data is fed into Forecast Agent application,which calculates Option Price Forecast making multiple price simulations in parallel.

4.7.2 Forecast Calculation Agent

The Core of the system is Forecast Calculation Agent. This is the program, which performs Monte-Carlo simulation of underlying assets prices for further Option price calculation. The program implements the model, described in Section 4.3.

Ptototype.Common is the library, which contains the implementation of the mathematical model, needed for performing the Monte-Carlo simulation. Model is implemented as a set of user defined types with a helper class, containing needed calculation method. Model type diagram is given in Figure 4.10

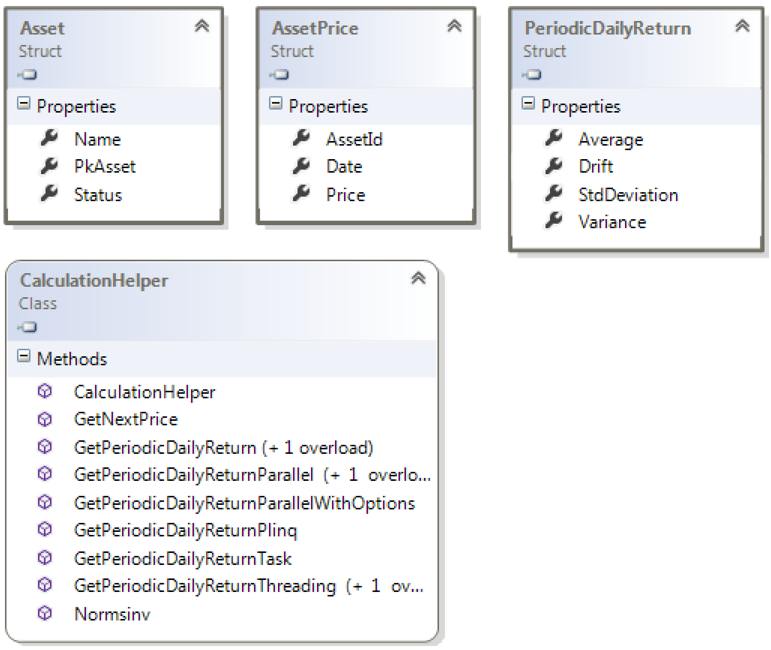

Asset
Struct
Properties
Name
PkAsset
Status
AssetPrice
Struct
Properties
AssetId
Date
Price
PeriodicDailyReturn
Struct
Properties
Average
Drift
StdDeviation
Variance
CalculationHelper
Class
Methods
CalculationHelper
GetNextPrice
GetPeriodicDailyReturn (+ 1 overload)
GetPeriodicDailyReturnParallel (+ 1 overlo...
GetPeriodicDailyReturnParallelWithOptions
GetPeriodicDailyReturnPlinq
GetPeriodicDailyReturnTask
GetPeriodicDailyReturnThreading (+ 1 ov...
Normsinv

Fig. 4.10: Model types

Prototype.SerialAgent is a serial implementation of the selected model calculation, used for creating the first working calculation prototype.

During the implementation of Prototype.ParallelAgent several approaches of multi-threading programming provided by .NET framework were tried out. The method which showed the best performance was used as a basic programming approach.

To have an idea, what parallel method of parallelism to prefer, one method from the mathematical model implementation was selected and different parallel versions of this method were implemented. Different parallel implementations were examined to performance. Method serial version is given below:

```
/// <summary>
/// Periodic Daily Return calculation
/// </summary>
/// <param name="closingPrices"></param>
/// <returns></returns>
public static PeriodicDailyReturn
   GetPeriodicDailyReturn(decimal[] closingPrices)
{
    ConcurrentBag<double> decimals = new ConcurrentBag<double>();

    for (int i = 0; i < closingPrices.Length - 1; i++)
    {
        double result = Math.Log((double)(closingPrices[i] /
            closingPrices[i + 1]), Math.E);
        decimals.Add(result);
    }

    PeriodicDailyReturn retParam = PeriodicDailyReturn(decimals);
    return retParam;
}
```

Serial version was made for general purposes. After that four more different version of that method were created for the parallelism examination. All methods were examined with mock data set, containing 50000 elements, on a Workstation, described in Table 4.13.

Tab. 4.14: Periodic Daily Return Calculation with 50000 elements

Parallel Option	**Time Spent (msec)**
No Parallelism	56
Managed Threading	143447
Explicit Task Parallelism	44
PLINQ	27
Implicit Task Parallelism (Parallel.For)	14

Due to the heavy-weight process of creating Thread objects in Managed Threading approach, this way is not considered as an appropriate way for the current prototype. The rest of the methods were examined again with datasets of different sizes starting from 1000 up to 150000000. Figure 4.11 shows the trend of performance of the same code using different parallelism ways.

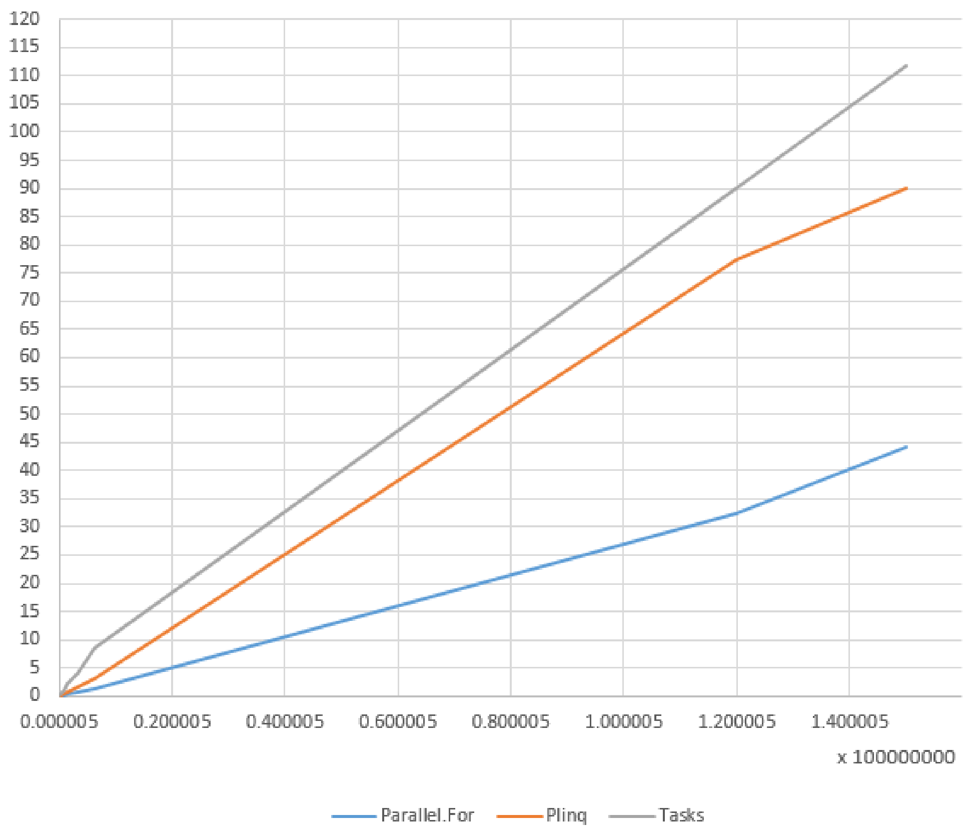


Fig. 4.11: Performance comparison

The last test results are given with exact numbers in Table 4.15.

Tab. 4.15: Periodic Daily Return Calculation with 150000000 elements

Parallel Option	**Time Spent (msec)**
Explicit Task Parallelism	111728
PLINQ	89939
Implicit Task Parallelism (Parallel.For)	44176

Testing was done using Scan primitive with mock data. Parallel.For performed in the best way. All tests were done at the Windows 7 Enterprise 64 bit operation system. The implementation of the method with a best performance was selected as a general approach for the Prototype.ParallelAgent creation.

```
Parallel.For(0, closingPrices.Length - 1, i =>
{
  int tempCount = i;
  double result = Math.Log((double)(closingPrices[tempCount] /
     closingPrices[tempCount + 1]), Math.E);
  decimals.Add(result);
});
```

4.7.3 Forecast Agent Sending and Triggering

Forecast Triggering Subsystem and Forecast Agent Sending Subsystem in current prototype are implemented in one software application. The name of the application is Prototype.Trigger and the Class diagram of the prototyped system is given in Figure 4.12

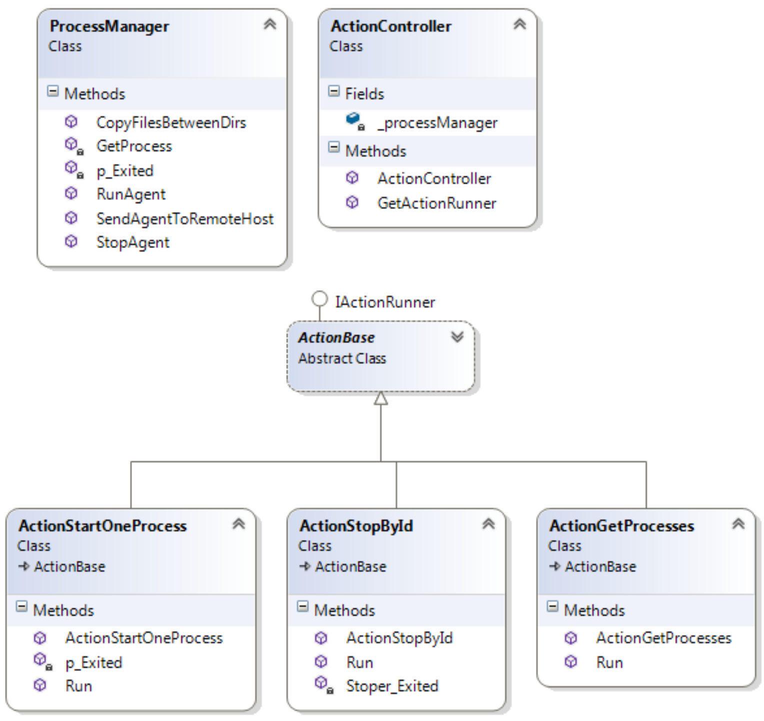


Fig. 4.12: Forecast Agent Sending and Triggering Application

One of the goals of this work is to demonstrate an approach of Crowd Computing during the creating of the system prototype. By assumption system operates at four available servers and in addition to that uses the workstations of the employees, available on the local area network. In .NET platform computers can be discovered using classes, providing functionality of Active Directory reading. Sample code is given in the the listing below.

```
PrincipalContext context = new
   PrincipalContext(ContextType.Domain, "domainname");
ComputerPrincipal compPrincipal = new ComputerPrincipal(context);
compPrincipal.Name = "*";

PrincipalSearcher principalSearch = new PrincipalSearcher();
principalSearch.QueryFilter = compPrincipal;
PrincipalSearchResult<Principal> result =
   principalSearch.FindAll();
```

With an appropriate set of rights Forecast program can be sent to discovered machines. Common Shared resource of Microsoft Windows operational system or in case of having administrator right pattern \computername\drive name$ can be used to access remote drive. Simple File.Copy can be used to transfer file from one computer to another. That principle can be used to create Private Crowd in local area network.

The part of the triggering subsystem, which allows to start, stop and monitor agents is implemented in the same Prototype.Trigger application. After the Forecast Agent is sent to remote host, it's managing can be done in different ways, for example:

- WMI
- Telnet
- psAnywhere

In this particular prototype Windows Sysinternals [54] utilities are used for saving the time. Prototype.Trigger application includes remote process management assemblies as a resources for managing of remote Forecast Agents.

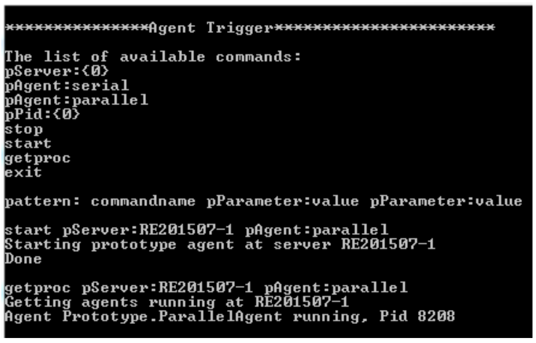
Figure 4.7 shows the class diagram for Forecast Agents control functionality – One common action base interface is defined and several implementing classes perform the job. Start, Stop and Monitor is implemented by wrapping the respective Sysinternals tool calls to the Prototype.Trigger application. Implementers produce parametrized calls and Process Manager communicates with remote hosts through the Sysinternals toolkit. The code sample of starting the Forecast Agents are given below.

```
public Process RunAgent(string agentParams)
{
  Process p = GetProcess("psexec", agentParams);
  p.EnableRaisingEvents = true;
  return p;
}
```

As it can be seen from the code samples above, Prototype.Trigger operates with .NET Process class. The method which return runnable process is described in the next code sample.

```
private static Process GetProcess(string appname, string
    arguments)
{
  Process p = new Process
    {
      StartInfo =
            {
              CreateNoWindow = true,
              FileName = string.Format(@"Resources\{0}.exe",
                  appname),
              Arguments = arguments
            };
      };
    return p;
}
```

Command pattern to operate the Prototype.Trigger is *commandname pParameter* : *value pParameter* : *value*. Parameters define how to start the system. Figure 4.13 demonstrates sending of the Forecast Agent to the available personal computer in a local area network.



```
***************Agent Trigger***********************

The list of available commands:
pServer:{0}
pAgent:serial
pAgent:parallel
pPid:{0}
stop
start
getproc
exit

pattern: commandname pParameter:value pParameter:value

start pServer:RE201507-1 pAgent:parallel
Starting prototype agent at server RE201507-1
Done

getproc pServer:RE201507-1 pAgent:parallel
Getting agents running at RE201507-1
Agent Prototype.ParallelAgent running, Pid 8208
```

Fig. 4.13: Prototype.Trigger application

By the moment of writing available commands parameters are:

- start, stop, monitor – start, stop or monitor Forecast Agent
- pServer:0 – the name of the server, where Forecast Agent is sent

- pAgent:parallel /pAgent:serial – agent type selection
- pPid:0 – running agent process id
- exit – exit Prototype.Trigger

5. EVALUATION OF APPROACH

This chapter briefly describes the results of testing of the created cluster computing system. During the testing of the whole system, a Monte-Carlo simulation of an Option Price Forecast Calculation was done using available computational resources, managed from one common place. As an input for the testing, 2000 assets were used. Each asset contains a price time series from the year 1960, which makes 20440 days at the time of writing. The chapter contains some notes about system benefits, weaknesses and technical constraints.

5.1 Performance Benchmarks

A prototype, made on the basis of the selected method, was examined in several configurations, starting from one single personal computer, and ending with a cluster consisting of 11 computers. The goal of the examination was to find out the speedup and possible bottlenecks of the system. The Option Price Forecast calculation was performed several times, and the results were documented and evaluated.

CPU involvement. The chosen approach to parallelization (Parallel.For) performed well in a multi core environment with all available CPU types. The goal of the selected parallelization approach was to make the program keep CPU busy, in that sense .NET standard way of implicit dynamic parallelism, used in Parallel.For construction, gave the best results. Figure 5.1 presents the results of two CPU involvement on a multi-processor system, which consists of two Intel Xeon X5660 processors [55], having six cores and twelve processor threads.

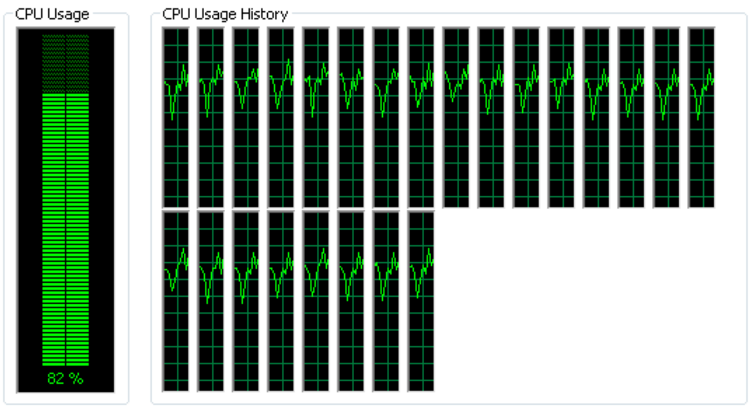


Fig. 5.1: Prototype.For testing

In comparison to managed threading with a manual thread start, shown in Figure 5.2, Parallel.For showed undisputed performance and efficiency. The best performance with Parallel.For was obtained using the settings for a maximum degree of parallelism multiple to CPU thread count.

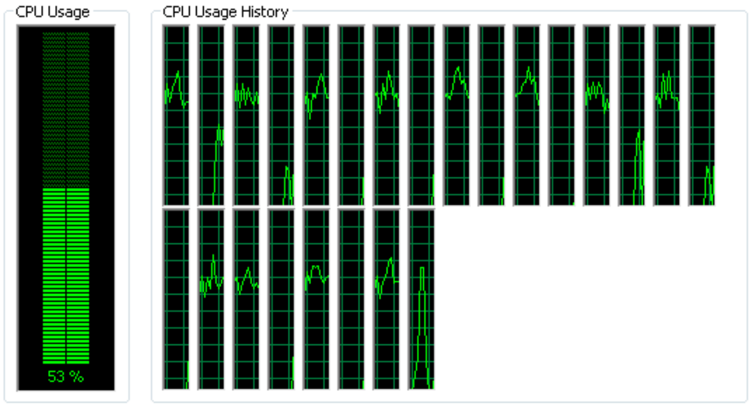


Fig. 5.2: Managed threading

Cluster computing. Option price forecast calculation was performed by a set of different computers. In that experiment the main idea was to see how the speedup of the system would be related to the increase in the computational nodes count. All the tests were done with one common database server MS SQL 2012, located on a standalone workstation. The details of the hardware used in testing are in Table 4.13.

The first test was done with an Intel Intel® Core^{TM} i7-4810MQ [56] CPU, equipped with 4 cores and 8 processor threads, which was connected to a local area network via a 1000MB/s network connection, which corresponds to the details of Workstation. The test was then repeated several times with the following hardware set:

- Server 1 and Server 2
- Server 1, Server 2 and Server 3
- Server 1, Server 2, Server 3 and Server 4
- Server 1, Server 2, Server 3, Server 4 and Personal Computer 1

- Server 1, Server 2, Server 3, Server 4 and two computers, similar to Typical Personal Computer
- Server 1, Server 2, Server, Server 4 and three computers, similar to Typical Personal Computer

The results of the tests are given in Table 5.1. Execution time is given in seconds

Tab. 5.1: Test Results

Computers count	**Run Time (sec)**
1	7420
2	3168
3	2442
4	2166
5	1623
6	1454
7	1245
8	1178
9	1145
10	1012
11	1011

Diagram in Figure 5.3 shows the decrease of the calculation time with an increase on the number of the computation nodes. The significant difference between the first test and others is caused by using different hardware.

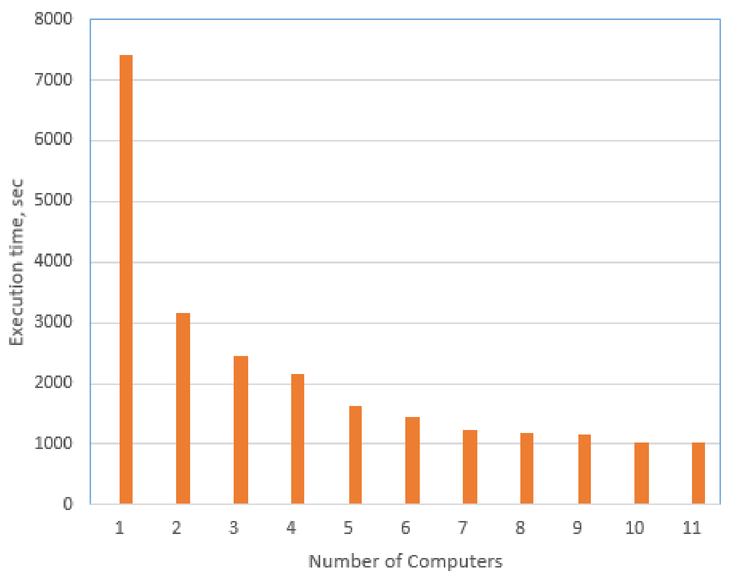


Fig. 5.3: Calculation speed

Figure 5.4 shows the relative speedup of the whole system after adding of computational resources. The performance increase is almost linear, however, the overall relative growth in performance depends on the type of hardware. Adding the weak node to the computational cluster can cause a decrease in speedup or even a slow down of the computation speed. This situation is visible in the Figure 5.4 between 4 and 5th test and after test 8.

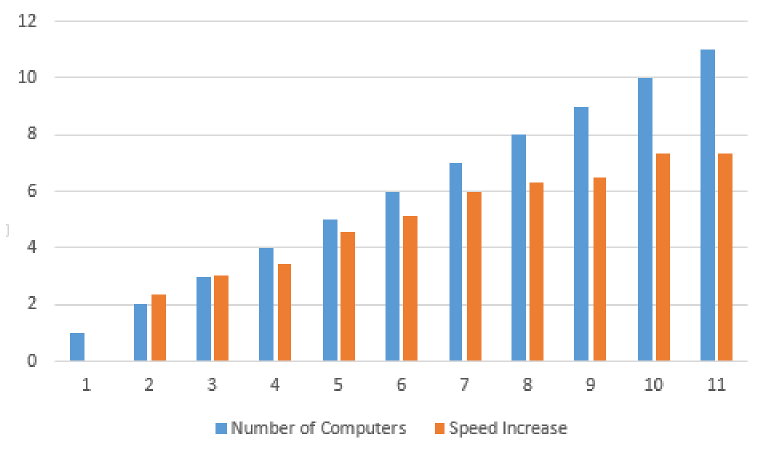


Fig. 5.4: Speed Increase

The presented test results lead to the conclusion that the prototyped system has a saturation point – the maximum number of resources that generate an increase in computation speed. In the case of the available software, the critical number was 8 computational resources. Further expansion of the cluster leads to a decrease of the processing speed and degradation of the system's overall performance. The reason for this are the increasing infrastructure costs – more time is needed to transfer data to the computation nodes and one single instance of MS SQL servers handles more requests to the procedure, which locks a primary data table for all threads until one thread reads the data.

Although the given system consists of different computational resources and includes different levels of parallelism for each participating calculation resource, the results of the test reflect Amdahl's Law [57], which describes a relation between the level of parallelism and the performance increase.

5.2 Benefits

The described approach, designed system and implemented prototype all together represent a very easy way of increasing calculation performance using a self-tailored computing cluster. Every company has at least several personal computers, which often stay idle. Using such system, it is possible to load all resources. In terms of pricing, if the company has four computers with four cores each and combines them together it can get a system with sixteen cores, working in data-parallel mode for free. The same option using Azure on the basis of a General Purpose computing platform with a twelve hour per day set up would cost around $530 [58] per month.

Short calculation of one existing system, used for Crude Oil price forecasting:

Average system daily time, min	726
CPU Cores Count in a system	36
CPU Threads in a system	72
2 * 16 cores Intel® Xeon® E5 CPU VMs	$2,322

The price is given without database server instance, in case of the need for Cloud MS SQL server, price would be appended with additional $1,399 per month for the premium MS SQL server instance with 500 GB storage. That shows clear benefit of described system in comparison to Cloud computing in the long – term perspective.

5.3 Tradeoffs

The given approach demonstrates a set of problems that require quite a large effort to solve. First, the system does not have a load balancer, which would ensure that the load is distributed equally between all nodes in the cluster. This leads to the weakest node of the cluster becoming a barrier for the whole system. Load balancing could require an additional layer, which would make the system more difficult to implement, install and maintain.

6. CONCLUSION

In accordance with the objectives of this work, the principles of pricing of financial derivatives and a comparison between the three existing methods have been studied. Among the existing methods of the financial derivatives pricing the method, which seemed to be the most interesting in terms of parallel and distributed computing was selected. Based on the selected derivative contract pricing methodology, a distributed system was designed. The system followed the principles of Crowd computing and was built in parallel and distributed programming patterns. The prototype of the designed system was created using the Microsoft .NET platform and a Windows ecosystem. Several prototype tests were made, where key points of testing were performance, scalability and efficient CPU usage. Despite some limitations in the prototype of the designed system, the testing results allowed the following conclusions to be made:

- pricing models examined using a Monte-Carlo approach are effective in assessing the value of derivative contracts that contain a large number of underlying assets.

- it is is possible to get an almost linear increase in the performance of a pricing system using the available personal computers and the principles of parallel and distributed computing.

- it is very simple to manufacture and set up an easily scalable and dynamically extendable computing cluster, which can replace or temporarily postpone the need for the expensive acquisition of a cloud-based or other specific computational resources.

The prototype showed some bottlenecks: the speed provided by the system is dependent on the speed of its weakest node. System performance depends on how quickly data is exchanged between the individual nodes. System design was carried out to be as abstract as possible, and as a result, was obtained a cross-platform design, which can be used for different data-parallel computations.

6.1 Future work

Further improvement of the system presented in this thesis should include the following parts:

- The system should contain a layer for initial data distribution in order to avoid bottlenecks on the reading of the initial data and writing out the calculation results.
- The system should have a load balancer in order to avoid a situation when one weak computation resource added to the cluster influences the performance of the whole system.
- The prototype could be ported to other languages and examined in other platforms, such as Unix-based operation systems and mobile platforms.

BIBLIOGRAPHY

[1] "Derivatives." http://www.investopedia.com/terms/d/derivative.asp. Accessed: 2016-05-04.

[2] F. Black and M. Scholes, "The pricing of options and corporate liabilities," in *The Journal of Political Economy, Vol. 81, No. 3*, pp. 637–654, 1973.

[3] S. Benninga and Z. Wiener, "The binomial option pricing model," in *Vol. 6 No. 3 1997 Mathematica in Education and Research publication*, pp. 1–9, 1997.

[4] "Option Basics : What are the options." http://www.investopedia.com/university/options/option.asp. Accessed: 2016-01-30.

[5] H. Moritsch, "Computational problems in finance," in *High Performance Computing in Finance—On the Parallel Implementation of Pricing and Optimization Models*, pp. 53–50, 2006.

[6] H. Moritsch, "Solution procedures," in *High Performance Computing in Finance—On the Parallel Implementation of Pricing and Optimization Models*, pp. 52–62, 2006.

[7] "Call option." http://www.investopedia.com/terms/c/calloption.asp. Accessed: 2016-01-30.

[8] "Put Option." http://www.investopedia.com/terms/p/putoption.asp. Accessed: 2016-01-30.

[9] "Strike Price." http://www.investopedia.com/terms/s/strikeprice.asp. Accessed: 2016-01-30.

[10] P. Glasserman, "Principles of monte carlo," in *Monte Carlo Methods in Financial Engineering*, pp. 1–19, 2003.

[11] "Risk-Free Rate Of Return." http://www.investopedia.com/terms/r/risk-freerate.asp. Accessed: 2016-01-30.

[12] "What is Parallel Computing." http://csis.pace.edu/~marchese/SE765/L0/L0b.htm. Accessed: 2016-04-04.

[13] B. Barney, "Why use parallel computing?," in *Introduction to Parallel Computing*, 2014.

[14] B. Barney, "Flynns taxonomy," in *Introduction to Parallel Computing*, 2014.

[15] M. O. Tokhi, M. A. Hossain, and M. H. Shaheed, "Classification based on memory arrangement and communication among pes," in *Parallel computing for real time signal processing and control*, pp. 30–31, 2003.

[16] M. O. Tokhi, M. A. Hossain, and M. H. Shaheed, "Shared memory multiprocessor," in *Parallel computing for real time signal processing and control*, p. 30, 2003.

[17] M. O. Tokhi, M. A. Hossain, and M. H. Shaheed, "Classification based on characteristic nature of processing elements," in *Parallel computing for real time signal processing and control*, p. 42, 2003.

[18] M. O. Tokhi, M. A. Hossain, and M. H. Shaheed, "Classification based on characteristic nature of processing elements," in *Parallel computing for real time signal processing and control*, p. 40, 2003.

[19] M. O. Tokhi, M. A. Hossain, and M. H. Shaheed, "Classification based on characteristic nature of processing elements," in *Parallel computing for real time signal processing and control*, p. 42, 2003.

[20] M. Forum, "Introduction to mpi," in *MPI, Message-Passing Interface Standard*, pp. 1–9, 2012.

[21] Springer, "Brent's theorem," in *Encyclopedia of Parallel Computing, Volume 4*, pp. 183–185, 2012.

[22] M. D. McCool, "Parallel computation patterns," in *Structured Parallel Programming with Deterministic Patterns*, 2010.

[23] M. van Steen, "Types of distibuted systems," in *Distibuted Systems: priciples and paradigsm*, pp. 1–14, 2012.

[24] M. van Steen, "Cluster computer system," in *Distibuted Systems: priciples and paradigsm*, pp. 17–18, 2012.

[25] M. van Steen, "Grid computer system," in *Distibuted Systems: priciples and paradigsm*, pp. 18–19, 2012.

[26] L. Sterling and K. Taveter, "From individual agents to multiagent systems," in *The Art of Agent-Oriented Modeling*, pp. 10–14, 2009.

[27] L. Sterling and K. Taveter, "Agents and activities," in *The Art of Agent-Oriented Modeling*, pp. 36–40, 2009.

[28] "Carnegie Mellon University." `http://www.cmu.edu/`. Accessed: 2016-05-04.

[29] "Apple website." `http://www.apple.com/`. Accessed: 2016-05-04.

[30] "Daimler AG." `https://www.daimler.com/en/`. Accessed: 2016-05-04.

[31] "HP." `http://www.hp.com/`. Accessed: 2016-05-04.

[32] "IBM." `http://www.ibm.com/us-en/`. Accessed: 2016-05-04.

[33] "Microsoft." `http://www.microsoft.com/et-ee/`. Accessed: 2016-05-04.

[34] "Oracle." `http://www.oracle.com/index.html`. Accessed: 2016-05-04.

[35] F. G.-L. de Guevara, "Towards an integrated crowdsourcing definition," in *Journal of Information Science*, pp. 1–14, 2012.

[36] "WorkFusion." `https://www.crunchbase.com/organization/crowdcomputing-systems#/entity`. Accessed: 2016-01-30.

[37] "Cloudfactory." `http://www.cloudfactory.com/`. Accessed: 2016-01-30.

[38] "PS3cluster." `http://moss.csc.ncsu.edu/~mueller/cluster/ps3/`. Accessed: 2016-01-30.

[39] "cloudfactory." `http://moss.csc.ncsu.edu/~mueller/cluster/ps3/`. Accessed: 2016-01-30.

[40] M. van Steen, "Transaction processing systems," in *Distibuted Systems: priciples and paradigsm*, pp. 20–23, 2012.

[41] "Rystad Energy." `http://www.rystadenergy.com/`. Accessed: 2016-04-04.

[42] C. de Schryver, I. Shcherbakov, F. Kienle, and N. Wehn, "An energy efficient fpga accelerator for monte carlo option pricing with the heston model," in *An Energy Efficient FPGA Accelerator for Monte Carlo Option Pricing with the Heston Model*, 2011.

[43] "PicsouGrid." `http://www-sop.inria.fr/oasis/personnel/Ian.Stokes-Rees/projects/PicsouGrid/docs/javadoc/picsou/grid/PicsouGrid.html`. Accessed: 2016-04-04.

[44] M. Research, "Dryad basics," in *Dryad and DryadLINQ: An Introduction*, pp. 1–14, 2009.

[45] "Microsoft drops Dryad." `http://www.informationweek.com/software/information-management/microsoft-ditches-dryad-focuses-on-hadoop/d/d-id/1101390?` Accessed: 2016-01-30.

[46] "Nokia Disco." `http://disco.readthedocs.org/en/latest/intro.html`. Accessed: 2016-01-30.

[47] "BOINC." `http://boinc.berkeley.edu/`. Accessed: 2016-04-04.

[48] "SETI[AT]HOME." `http://setiathome.ssl.berkeley.edu/`. Accessed: 2016-04-04.

[49] F. C. L. Veiga and P. Ferreira, "Boinc-mr architecture," in *BOINC-MR: MapReduce in a Volunteer Environment*, pp. 3–4, 2012.

[50] J. Dean and S. Ghemawat, "Introduction," in *MapReduce: Simplified Data Processing on Large Clusters*, pp. 1–11, 2002.

[51] "Yahoo Finance." `http://finance.yahoo.com/`. Accessed: 2016-03-10.

[52] "Microsoft .NET." `https://www.microsoft.com/net/default.aspx`. Accessed: 2016-03-10.

[53] "MS Sql Server 2012." `https://www.microsoft.com/en-us/download/details.aspx?id=35575`. Accessed: 2016-03-10.

[54] "Windows Sysinternals." `https://technet.microsoft.com/en-us/sysinternals/psexec.aspx`. Accessed: 2016-03-30.

[55] "Xeon 5660 specifications." `http://ark.intel.com/products/47921/Intel-Xeon-Processor-X5660-12M-Cache-2_80-GHz-6_40-GTs-Intel-QPI`. Accessed: 2016-01-30.

[56] "Intel® Core^{TM} i7-4810MQ Processor specifications." `http://ark.intel.com/products/78937/Intel-Core-i7-4810MQ-Processor-6M-Cache-up-to-3_80-GHz`. Accessed: 2016-01-30.

[57] J. Dean and S. Ghemawat, "Amdahl's law in the multicore era,"

[58] "Microsoft Azure." `https://azure.microsoft.com/en-us/pricing/details/virtual-machines`. Accessed: 2016-04-04.