Morgan Stanley

Introduction to Computational Finance

30657

Norbert Fogarasi
Managing Director, Morgan Stanley

Financial Information Systems
BME-VIK

Computational Finance

- Computational finance is a branch of applied computer science that deals with problems of practical interest in finance.
- It is a relatively new discipline whose birth can be traced back to the work of Harry Markowitz in the 1950s and since then has provided many problems of deep algorithmic interest.
- Presently, it encompasses various new numerical methods in the fields of optimization, statistics, and stochastic processes developed for different financial applications.
  - MPT / Portfolio selection
  - Algorithmic Trading, HFT
  - Parallel computing techniques (GPGPU, FPGA)
  - Monte Carlo techniques

Pricing vs. Risk vs. Trading

- Pricing: Finding the “fair” value of a financial product based on assumptions on the market dynamics
  - Arbitrage free pricing
  - Risk neutral pricing

- Risk: Finding the sensitivity of a price to various variables
  - Market risk: sensitivity to market variables. Equivalent to mathematical partial derivative. Often computed by “bumping” the input parameter by a small amount and taking the pricing difference.
  - Credit risk: sensitivity to changes in the credit worthiness of counterparties or trade underlyings
  - Operational risk: sensitivity to operational incidents (eg. cyber, mistakes etc)
  - Liquidity risk: sensitivity to disruptions in market dynamics

- Trading: finding an algorithm or strategy for buying/selling instruments

Numerical methods

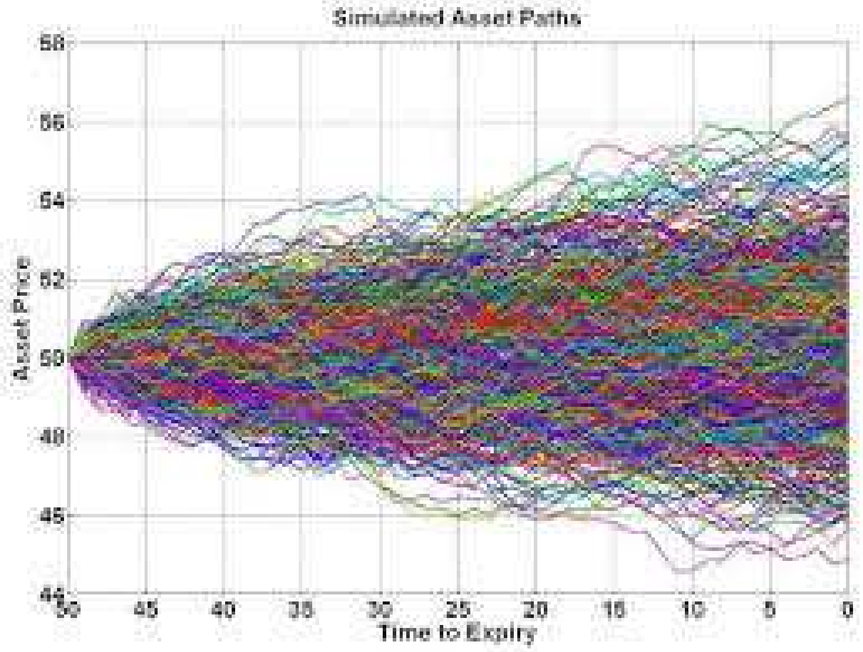
- Pricing can be closed form (eg. Black-Scholes, simple bond pricing etc)
  - Given a set of inputs, a well-defined set of deterministic operations are applied to obtain a price

- Discrete grid/tree based pricing models
  - Numerical approximation
  - Discretization
  - Can be iterative
  - Eg. binary tree based option pricing covered last time, finite difference methods
- Simulation based pricing
  - Started in the 1960's for corporate finance, 1977 for options pricing (Phelim Boyle)
  - Simulating multiple potential outcomes of undertain market variebles to determine the distribution of the prices over the range of resultant outcomes.

Monte Carlo Simulations

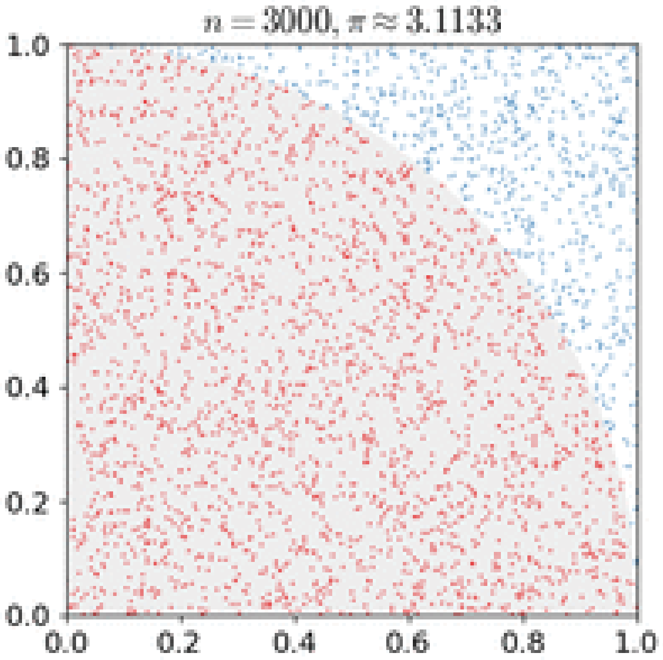
- Based on the assumed dynamic of the unerlying, many paths can be generated and the value of the derivative evaluated on each.
- Monte Carlo methods converge to the solution more quickly than numerical methods, require less memory and are easier to program.
- For simpler situations, however, simulation is not the better solution because it is very time-consuming and computationally intensive.
- Monte Carlo methods can deal with derivatives which have path dependent payoffs.

Monte Carlo Method was discovered by Nicolas Metropolis in 1947



A Simple Monte Carlo Estimation

- Estimating the value of Pi/4:



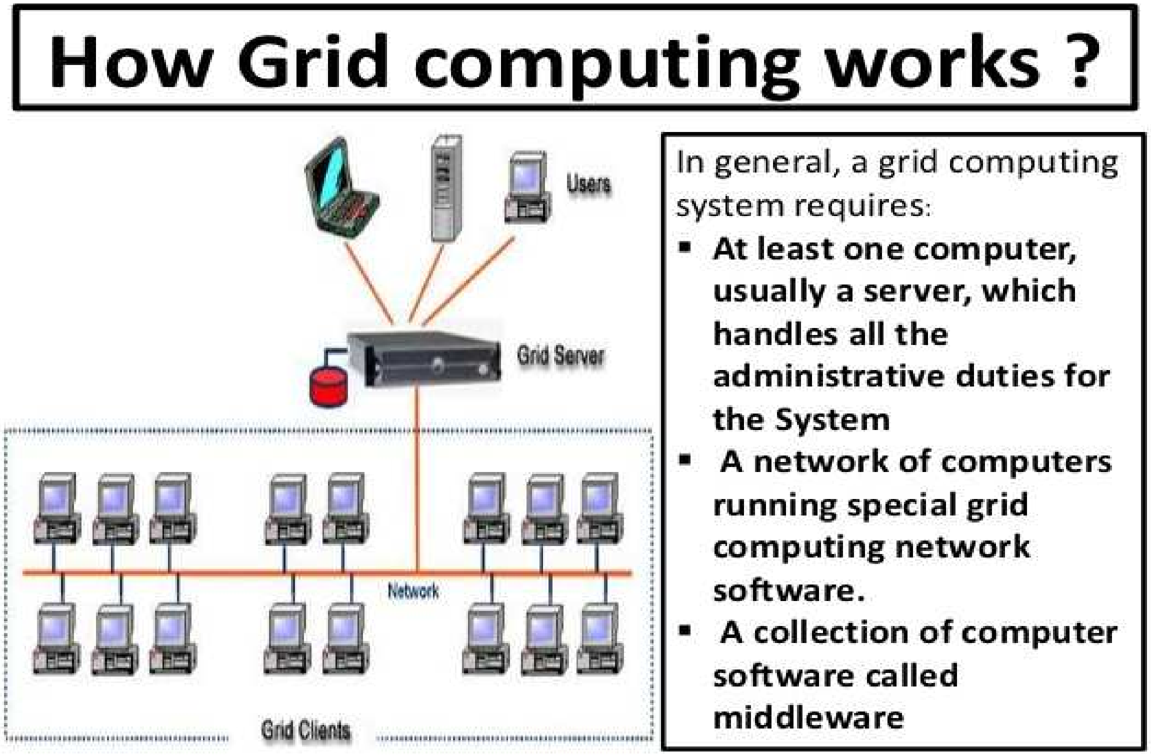
(Source: Wikipedia)

Advanced Topics in MC Simulations

- Estimates for the risks of an option i.e. the (mathematical) derivatives of option value with respect to input parameters, can be obtained by numerical differentiation.
- This can be a time-consuming process (an entire Monte Carlo run must be performed for each "bump" or small change in input parameters). Further, taking numerical derivatives tends to emphasize the error (or noise) in the Monte Carlo value - making it necessary to simulate with a large number of sample paths.
- Practitioners regard these points as a key problem with using Monte Carlo methods.
- Various variance reduction methods can be applied to standard MC to increase practical applicability (eg. importance sampling, quasi-random numbers etc)

Grid Computing

- **Grid computing** is the collection of computer resources from multiple locations to reach a common goal.
- The grid is a distributed system with non-interacting, independent tasks.



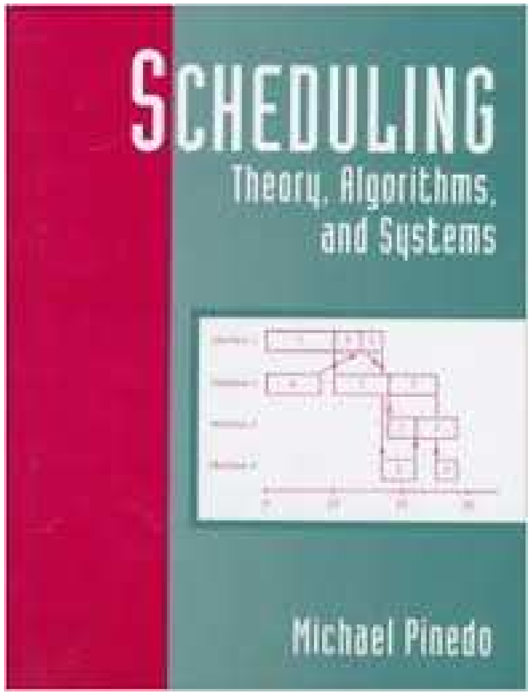
How Grid computing works ?

In general, a grid computing system requires:

- **At least one computer, usually a server, which handles all the administrative duties for the System**
- **A network of computers running special grid computing network software.**
- **A collection of computer software called middleware**

Scheduling Theory

- **Scheduling Theory** is a branch of applied mathematics concerned with finding optimal ordering and coordination in time of tasks.
- Depending on the number and types of machines, tasks and other environmental factors, we can determine how difficult the scheduling task is
- If the scheduling is very difficult, we may have to look for approximate solution which is quick to find.
- Minimizing the Total Weighted Tardiness of jobs on Identical machines is an NP-hard problem.



Scheduling Tasks on a Grid

- Complex portfolios are evaluated and risk managed using Monte-Carlo simulations at many financial institutions
- Future trajectories of market variables are simulated and portfolio value/risk is evaluated on each trajectory, then a weighted average is used
- Each night a changed portfolio needs to be evaluated/risk managed with new market data/model parameters
- Need a quick way to schedule 10000’s of jobs on 10000’s of machines in a near optimal way
- Why? $10M/year spend on hardware, timely response to clients and regulators regarding portfolio values and risk.

Mathematical Problem Formulation

- Scheduling jobs on a finite number (V) of identical processors under constraints on the completion times
- Given n users/jobs of sizes $\mathbf{x} = \{x_1, x_2, ..., x_n\} \in \mathbf{N}^n$
- Cutoff times $\mathbf{K} = \{K_1, K_2, ..., K_n\} \in \mathbf{N}^n$
- Weights/priorities $w = \{w_1, w_2, ..., w_n\} \in \mathbf{R}^n$
- Scheduling matrix: $\mathbf{C} \in \{0,1\}^{n \times m}$
- Where $\mathbf{C}_{i,j} = 1$ if job i is processed at time step j.
- Jobs can stop/restart on different machine (preemption)
- For example V=2, n=3, x={2,3,1}, K={3,3,3}.

$$\mathbf{C} = \begin{array}{cc} \text{Time Steps} & \\ \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} & \text{Jobs} \end{array}$$

Problem Formulation

- Define “Tardiness” as $T_i = \max(0, F_i - K_i)$
- where $F_i := \arg\max_j \{\mathbf{C}_{i,j} = 1\}$ is the finishing time of job i as per C.
- Minimizing Total Weighted Tardiness (TWT) is stated as

$$\mathbf{C}_{opt} := \arg\min_{\mathbf{C}} \sum_{i=1}^{N} w_i T_i$$

- Under the following constraints:

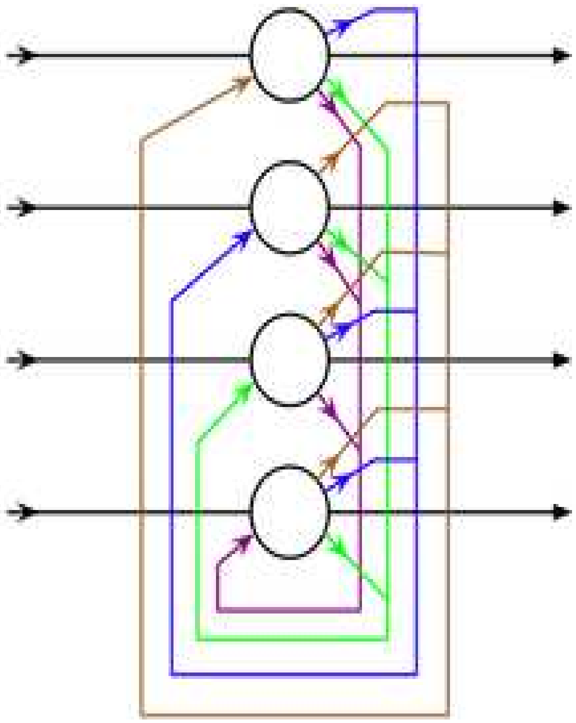
$$\sum_{j=1}^{L} C_{i,j} = x_i, \forall i = 1, \ldots, N \qquad \sum_{i=1}^{N} C_{i.j} \leq V, \forall j = 1, \ldots, L$$

- For example V=2, n=3, x={2,3,2}, K={3,3,3}, w={3,2,1} All jobs cannot complete before their cutoff times, but the optimal TWT solution is:

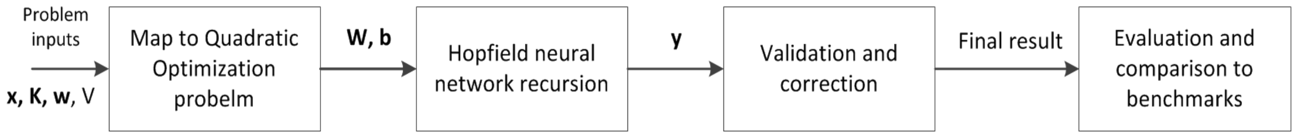
$$\mathbf{C} = \begin{array}{c} \text{Time Steps} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \text{Jobs}$$

Hopfield Neural Networks (HNN)

- A *Hopfield* network is a form of recurrent artificial *neural network* popularized by John *Hopfield* in 1982.
- Can be used to model human memory
- Very efficient to find a fast approximate solution for quadratic optimization problems



Novel Approach: TWT to QP



Problem inputs

x, K, w, V

Map to Quadratic Optimization probelm

W, b

Hopfield neural network recursion

y

Validation and correction

Final result

Evaluation and comparison to benchmarks

- HNN are a recursive Neural Network which are good for solving quadratic optimization problems in the form

$$f\left(\mathbf{y}\right) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} + \mathbf{b}^T\mathbf{y},$$

- Our task is to transform the TWT to a quadratic optimization problem.

$$\min \sum_{i=1}^{N} w_i \left(\sum_{j=K_i+1}^{L} C_{i,j} \right)$$

$$\forall i: \sum_{j=1}^{L} C_{i,j} = x_i \rightarrow \min_{\mathbf{C}} \sum_{i=1}^{N} \left(\left(\sum_{j=1}^{L} C_{i,j} \right) - x_i \right)^2$$

$$\forall j: \sum_{i=1}^{N} C_{i.j} = V \rightarrow \min \sum_{i=1}^{L} \left(\left(\sum_{j=1}^{N} C_{i,j} \right) - V \right)^2$$

Novel Approach: TWT to QP

Move constraints to objective function:

$$\min E(\mathbf{C}) = \min \alpha \sum_{j=1}^{J} \sum_{l=K_j}^{L} w_j C_{jl}^2 + \beta \sum_{i=1}^{N} \left(\left(\sum_{j=1}^{L} C_{i,j} \right) - x_i \right)^2 + \gamma \sum_{i=1}^{L} \left(\left(\sum_{j=1}^{N} C_{i,j} \right) - V \right)^2$$

- Each member of the above addition can be converted to quadratic Lyapunov form separately to bring the expression into the form

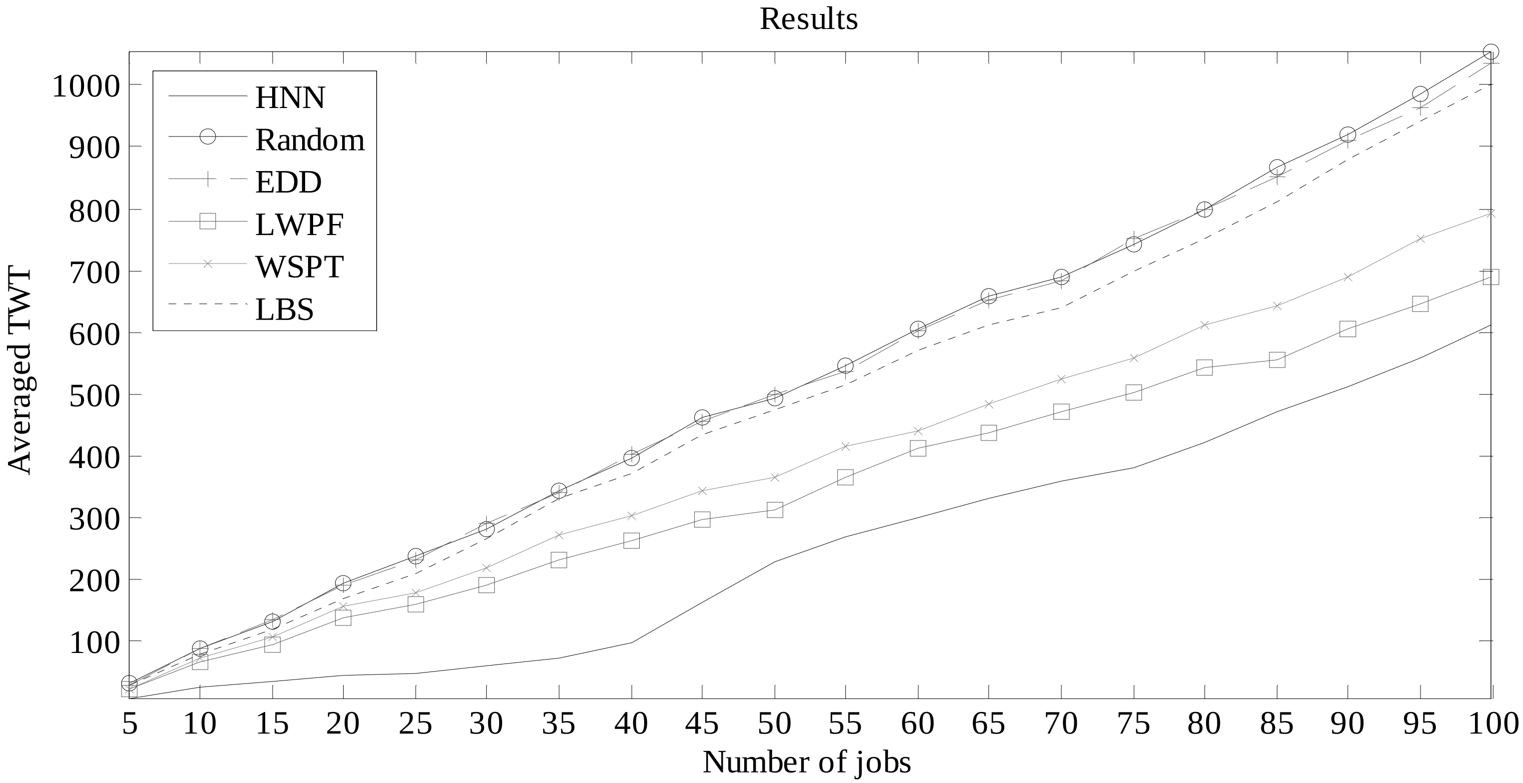
$$f(\mathbf{y}) = -\frac{1}{2} \mathbf{y}^T \mathbf{W} \mathbf{y} + \mathbf{b}^T \mathbf{y},$$

$$\mathbf{W} = \alpha \mathbf{W}_A + \beta \mathbf{W}_B + \gamma \mathbf{W}_C \in \mathbf{R}^{NL \times NL}$$

$$\mathbf{b} = \alpha \mathbf{b}_A + \beta \mathbf{b}_B + \gamma \mathbf{b}_C \in \mathbf{R}^{NL \times 1}$$

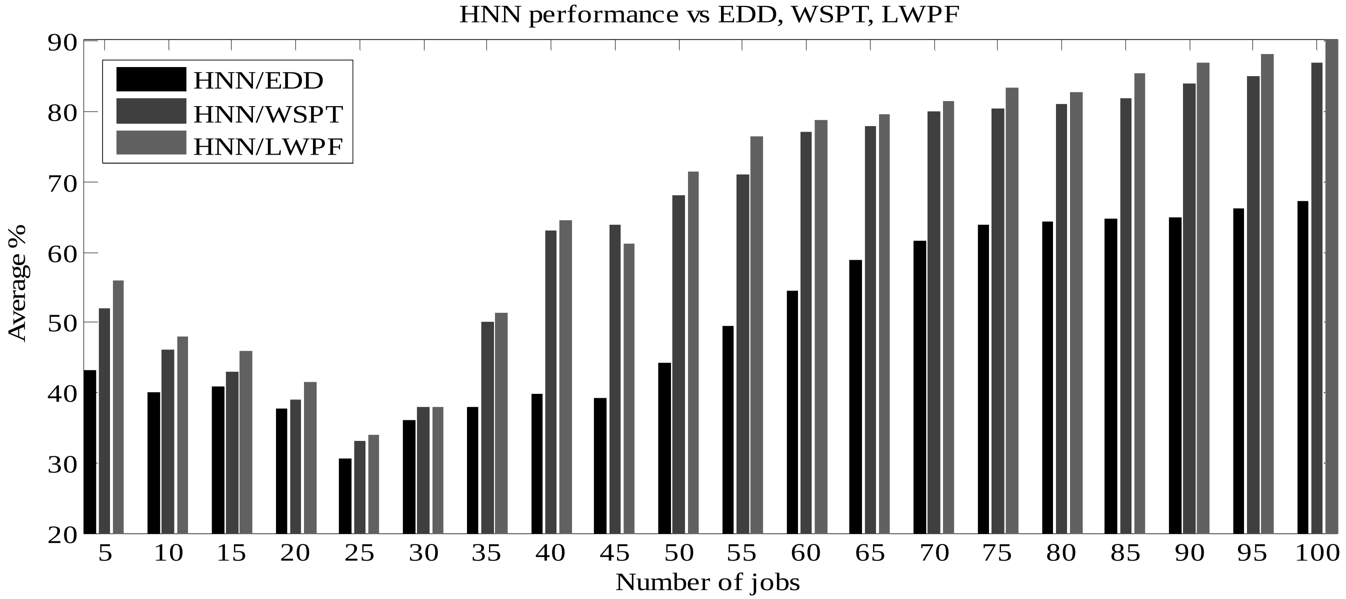
HNN outperforms other simple heuristics

For each problem size (# of jobs) 100 random problems were generated and the average TWT was computed and plotted



HNN outperforms other simple heuristics

Outperformance is consistent over a broad spectrum of problems over simple heuristics in literature (LWPF – Largest Weighted Process First, WSPT – Weighted Shortest Processing Time, EDD – Earliest Due Date)



Job size	5	10	15	20	30	40	50	75	100
% outperf	99.9	100	100	99.5	99.2	99.6	99.3	98.6	98.8