University of Texas at El Paso
DigitalCommons@UTEP

Departmental Technical Reports (CS) | Department of Computer Science

2-1-2012

Towards Optimizing Cloud Computing: An Example of Optimization under Uncertainty

Vladik Kreinovich
University of Texas at El Paso, vladik@utep.edu

Follow this and additional works at: http://digitalcommons.utep.edu/cs_techrep

Part of the Computer Engineering Commons

Comments:
Technical Report: UTEP-CS-11-54b
To appear in: Samee U. Khan, Lizhe Wang, and Albert Y. Zomaya (eds.), *Scalable Computing and Communications: Theory and Practice*, John Wiley & Sons.

Recommended Citation
Kreinovich, Vladik, "Towards Optimizing Cloud Computing: An Example of Optimization under Uncertainty" (2012). *Departmental Technical Reports (CS)*. Paper 634.
http://digitalcommons.utep.edu/cs_techrep/634

This Article is brought to you for free and open access by the Department of Computer Science at DigitalCommons@UTEP. It has been accepted for inclusion in Departmental Technical Reports (CS) by an authorized administrator of DigitalCommons@UTEP. For more information, please contact lweber@utep.edu.

CHAPTER 1

TOWARDS OPTIMIZING CLOUD COMPUTING: AN EXAMPLE OF OPTIMIZATION UNDER UNCERTAINTY

VLADIK KREINOVICH

Department of Computer Science, University of Texas at El Paso,
El Paso, TX 79968, USA, vladik@utep.edu

One of the most efficient way to store and process data is *cloud computing*. In cloud computing, instead of storing the data at the user-defined location (e.g., at the user's computer or at the centralized server), the computer system ("cloud") selects the location of the data storage that speeds up computations – by minimizing the (average) communication time. In this chapter, we provide an analytical solution to the corresponding optimization problem.

The demand for cloud computing is growing fast, and we expect that this demand – and thus, the size of the resulting cloud – will continue to grow. To avoid expensive frequent redesigns of the cloud, it is therefore desirable to make sure that the resulting servers will have enough capacity to satisfy future demand – and at the same time that we do not build in expensive extra capacity that will not be used in the predictable future. It is thus important to be able to predict the future demand for cloud computing – i.e., predict how the cloud will grow. In this chapter, we describe how to optimally predict the cloud growth.

Scalable Computing: Theory and Practice, 1st edition. By Samee U. Khan, Lizhe Wang, and Albert Y. Zomaya
Copyright © 2012 John Wiley & Sons, Inc.

1.1 CLOUD COMPUTING: WHY WE NEED IT AND HOW CAN WE MAKE IT MOST EFFICIENT

Why cloud computing. In many application areas (bioinformatics, geosciences, etc.) we need to process large amounts of data, which require fast computers and fast communication. Historically, there have been limits on the amount of the information that can be transmitted at a high speed, and these limits affected information processing.

A few decades ago, computer connections were relatively slow, so electronically transmitting a large portion of a database required a lot of time. It was, however, possible to transmit the results of the computations really fast. As a result, the best strategy to get fast answers to users' requests was to move all the data into a central location, close to the high performance computers for processing this data.

In the last decades, it became equally fast to move big portions of databases needed to answer a certain query. This enabled the users to switch to a *cyberinfrastructure* paradigm, when there is no longer a need for time-consuming moving of data to a central location: the data is stored where it was generated, and when needed, the corresponding data is moved to processing computers; see, e.g., [4, 6, 12, 14, 15] and references therein.

Nowadays, moving the whole databases becomes almost as fast as moving their portions, so there is no longer need to store the data where it was produced – it is possible to store the data where it will be best for future data processing. This idea underlies the paradigm of *cloud computing*.

What is the most efficient way of cloud computing. The main advantage of cloud computing is that, in comparison with the centralized computing and with the cyberinfrastructure-type computing, we can get answers to queries faster – by finding optimal placement of the servers that store and/or process the corresponding databases. So, in developing cloud computing schemes, it is important to be able to solve the corresponding optimization problems.

We have started solving these problems in [10]. In this chapter, we expand our previous results and provide a solution to the problem of the optimal server placement – and to the related optimization problems.

1.2 OPTIMAL SERVER PLACEMENT PROBLEM: FIRST APPROXIMATION

What we want and what we need. For each database (e.g., a database containing geophysical data), we usually know how many requests (queries) for data from this database come from different geographic locations x. These numbers of requests can be described by the *geographic (request) density* function $\rho_r(x)$ describing the number of requests per unit time and per unit area around the location x. We also usually know the number of duplicates D of this database that we can afford to store.

Our objective is to find the optimal locations of D servers storing these duplicates – to be more precise, locations that minimize the average response time. The desired locations can also be characterized by a density function – namely, by the *storage density* function $\rho_s(x)$ describing the number of copies per geographic region (i.e., per unit area in the vicinity of the location x).

Once a user issues a request, this request is communicated to one of the servers storing a copy of the database. This server performs the necessary computations, after which the result is communicated back to the user. The necessary computations are usually relatively fast – and the corresponding computation time does not depend on where the database is actually stored. So, to get the answers to the users as soon as possible, we need to minimize the communication time delay.

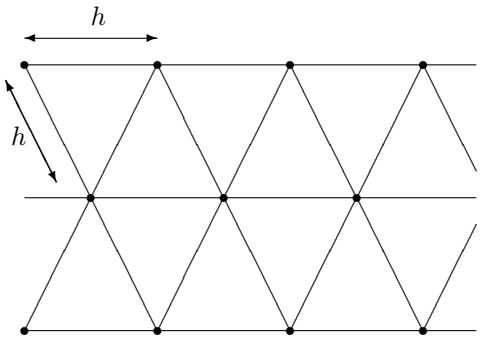
Thus, we *need to determine* the storage density functio $\rho_s(x)$ that minimizes the average communication delay.

First approximation model: main assumption. In the first approximation, we can measure the travel delay by the average travel distance. Under this approximation, minimizing the travel delay is equivalent to minimizing the average travel distance.

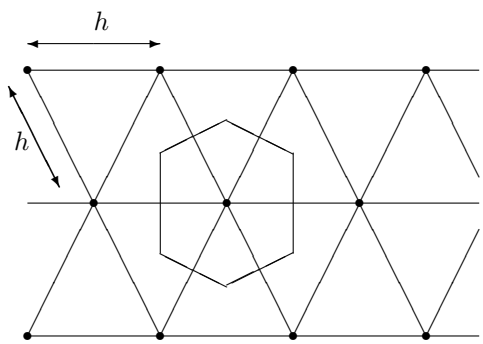
Derivation of the corresponding model. How can we describe this distance in terms of the density $\rho_s(x)$? When the density is constant, we want to place the servers in such a way that the largest distance r to a server is as small as possible. (Alternatively, if r is fixed, we want to minimize the number of servers for which every point is at a distance $\leq r$ from one of the servers.) In geometric terms, this means that every point on a plane belongs to a circle of radius r centered on one the servers – and thus, the whole plane is covered by such circles. Out of all such coverings, we want to find the covering with the smallest possible number of servers.

It is known that the smallest such number is provided by an equilateral triangle grid, i.e., a grid formed by equilateral triangles; see, e.g., [8, 9].

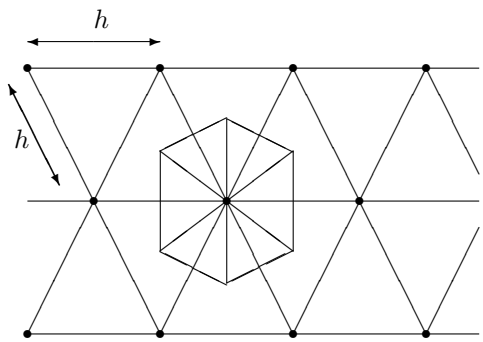
Let us assume that we have already selected the server density function $\rho_s(x)$. Within a small region of area A, we have $A \cdot \rho_s(x)$ servers. Thus, if we, e.g., place these servers on a grid with distance h between the two neighboring ones in each direction, we have:



For this placement, the set of all the points which are closest to a given server forms a hexagonal area:



This hexagonal area consists of 6 equilateral triangles with height $h/2$:



In each triangle, the height $h/2$ is related to the size s by the formula

$$\frac{h}{2} = s \cdot \sin(60^\circ) = s \cdot \frac{\sqrt{3}}{2},$$

hence

$$s = \frac{h}{\sqrt{3}} = h \cdot \frac{\sqrt{3}}{3}.$$

Thus, the area A_t of each triangle is equal to

$$A_t = \frac{1}{2} \cdot s \cdot \frac{h}{2} = \frac{1}{2} \cdot \frac{\sqrt{3}}{3} \cdot \frac{1}{2} \cdot h^2 = \frac{\sqrt{3}}{12} \cdot h^2.$$

So, the area A_s of the whole set is equal to 6 times the triangle area:

$$A_s = 6 \cdot A_t = \frac{\sqrt{3}}{2} \cdot h^2.$$

Each point from the region is the closest to one of the points from the server grid, so the region of area A is thus divided into $A \cdot \rho_s(x)$ (practically) disjoint sets of area $\frac{\sqrt{3}}{2} \cdot h^2$. So, the area of the region is equal to the sum of the areas of these sets:

$$A = (A \cdot \rho_s(x)) \cdot \frac{\sqrt{3}}{2} \cdot h^2.$$

Dividing both sides of this equality by A, we conclude that

$$1 = \rho_s(x) \cdot \frac{\sqrt{3}}{2} \cdot h^2,$$

and hence, that

$$h = \frac{c_0}{\sqrt{\rho_s(x)}},$$

where we denote

$$c_0 \stackrel{\text{def}}{=} \sqrt{\frac{2}{\sqrt{3}}}.$$

The largest distance r to a server is thus equal to

$$r = \frac{h}{2} = \frac{c_0}{2 \cdot \sqrt{\rho_s(x)}}.$$

The average distance $\overline{d}$ is proportional to r – since when we re-scale the picture, all the distances –including the average distance – increase proportionally. Since the distance r is proportional to $(\rho_s(x))^{-1/2}$, the average distance near the location x is thus also proportional to this same value: $\overline{d}(x) = \text{const} \cdot (\rho_s(x))^{-1/2}$ for some constant.

At each location x, we have $\sim \rho_r(x)$ requests. Thus, the total average distance – the value that we would like to minimize – is equal to $\int \overline{\rho}(x) \cdot \rho_r(x)\, dx$ and is, therefore, proportional to

$$\int (\rho_s(x))^{-1/2} \cdot \rho_r(x)\, dx.$$

So, minimizing the average distance is equivalent to minimizing the value of the above integral.

We want to find the server placement $\rho_s(x)$ that minimizes this integral under the constraint that the total number of server is D, i.e., that $\int \rho_s(x) = D$.

Resulting constraint optimization problem. Thus, we arrive at the following optimization problem:

- We know the density $\rho_r(x)$ and an integer D;
- under all possible functions $\rho_s(x)$ for which $\int \rho_s(x)\, dx = D$, we must find a function that minimizes the integral $\int (\rho_s(x))^{-1/2} \cdot \rho_r(x)\, dx$.

Solving the resulting constraint optimization problem. A standard way to solve a constraint optimization problem of optimizing a function $f(X)$ under the constraint $g(X) = 0$ is to use the Lagrange multiplier method, i.e., to apply unconstrained optimization to an auxiliary function $f(X) + \lambda \cdot g(X)$, where the parameter λ (called *Lagrange multiplier*) is selected in such a way so as to satisfy the constraint $g(X) = 0$.

With respect to our constraint optimization problem, this means that we need to select a density $\rho_s(x)$ that optimizes the following auxiliary expression:

$$\int (\rho_s(x))^{-1/2} \cdot \rho_r(x)\, dx + \lambda \cdot \left(\int \rho_s(x)\, dx - D \right).$$

Having an unknown function $\rho_s(x)$ means, in effect, that we have infinitely many unknown values $\rho(x)$ corresponding to different locations x. Optimum is attained when the derivative with respect to each variable is equal to 0. Differentiating the above expression with respect to each variable $\rho_s(x)$, and equating the result to 0, we get the equation

$$-\frac{1}{2} \cdot (\rho_s(x))^{-3/2} \cdot \rho_r(x) + \lambda = 0,$$

hence $\rho_s(x) = c \cdot (\rho_r(x))^{2/3}$ for some constant c.

The constant c can be determined from the constraint $\int \rho_s(x)\, dx = D$, i.e., that

$$\int c \cdot (\rho_r(x))^{2/3}\, dx = c \cdot \int (\rho_r(x))^{2/3}\, dx = D.$$

Thus,

$$c = \frac{D}{\int (\rho_r(x))^{2/3}\, dx},$$

and we arrive at the following solution.

Solution to the problem. Once we know the request density $\rho_r(x)$ and the total number of servers D that we can afford, the optimal server density $\rho_s(x)$ is equal to

$$\rho_s(x) = D \cdot \frac{(\rho_r(x))^{2/3}}{\int (\rho_r(y))^{2/3}\, dy}.$$

Discussion. In line with common sense, the optimal server density increases when the request density increases, i.e.:

- in locations that generate more requests, we place more servers, and
- in locations that generate fewer requests, we place fewer servers.

However, when the request density decreases, the server density decreases slower – because otherwise, if we took the server density simply proportional to the request density, the delays in areas with few users would have been huge.

It is worth mentioning that similar conclusions come from the analysis of a different – security-related – optimization problem, in which, instead of placing servers, we need to place sensors; see [9].

1.3 SERVER PLACEMENT IN CLOUD COMPUTING: TOWARDS A MORE REALISTIC MODEL

First idea. In the above first approximation, we only took into account the time that it takes to move the data to the user. This would be all if the database was not changing. In real life, databases need to be periodically updated. Updating also takes time. Thus, when we find the optimal placement of servers, we need to take into account not only expenses on moving the data to the users, but also the expenses of updating the information.

Towards a precise formulation of this idea. How do we estimate these expenses? In a small area, where the user distribution is approximately uniform, the servers are also uniformly distributed, i.e., they form a grid with distance $h = 2r$ between the two neighboring servers [8, 9]. Within a unit area, there are $\sim 1/r^2$ servers, and reaching each of them from one of its neighbors requires time proportional to the distance $\sim r$. The overall effort of updating all the servers can be obtained by multiplying the number of servers by an effort needed to update each server, and is thus proportional to $1/r^2 \cdot r \sim 1/r$. We already know that $r \sim (\rho_s(x))^{-1/2}$, thus, the cost of updating all the servers in the vicinity of a location x is proportional to $(\rho_s(x))^{1/2}$. The overall update cost can thus be obtained by integrating this value over the whole area. Thus, we arrive at the following problem.

Resulting optimization problem.

- We know the density $\rho_r(x)$, an integer D, and a constant C that is determined by the relative frequency of updates in comparison with frequency of normal use of the database;
- under all possible functions $\rho_s(x)$ for which $\int \rho_s(x)\,dx = D$, we must find a function that minimizes the expression

$$\int (\rho_s(x))^{-1/2} \cdot \rho_r(x)\,dx + \int C \cdot (\rho_s(x))^{1/2}\,dx.$$

Solving the problem. To solve the new optimization problem, we can similarly form the Lagrange multiplier expression

$$\int (\rho_s(x))^{-1/2} \cdot \rho_r(x)\,dx + \int C \cdot (\rho_s(x))^{1/2}\,dx + \lambda \cdot \left(\int \rho_s(x)\,dx - D\right),$$

differentiate it with respect to each unknown $\rho_s(x)$, and equate the resulting derivative to 0. As a result, we get an equation

$$-\frac{1}{2} \cdot (\rho_s(x))^{-3/2} \cdot \rho_r(x) + \frac{1}{2} \cdot C \cdot (\rho_s(x))^{-1/2} + \lambda = 0.$$

This is a cubic equation in terms of $(\rho_s(x))^{-1/2}$, so while it is easy to solve numerically, there is no simple analytical expression as in the first approximation case.

The resulting solution $\rho_s(x)$ depends on the choice of the Lagrange multiplier λ, i.e., in effect, we have $\rho_s(x) = \rho_s(x, \lambda)$. The value λ can be determined from the condition that $\int \rho_s(x, \lambda)\, dx = D$.

Second idea. The second idea is that usually, a service provides a time guarantee, so we should require that no matter where a user is located, the time for this user to get the desired information from the database should not exceed a certain value. In our model, this means that a distance r from the user to the nearest server should not exceed a certain given value r_0. Since $r \sim (\rho_s(x))^{-1/2}$, this means, in turn, that the server density should not decrease below a certain threshold ρ_0.

This is an additional constraint that we impose on $\rho_s(x)$. In the first approximation model, it means that instead of the formula $\rho_s(x) = c \cdot (\rho_r(x))^{2/3}$ – which could potentially lead to server densities below ρ_0 – we should have $\rho_s(x) = \max(c \cdot (\rho_r(x))^{2/3}, \rho_0)$.

The parameter c can be determined from the constraint

$$\int \rho_s(x)\, dx = \int \max(c \cdot (\rho_r(x))^{2/3}, \rho_0)\, dx = D.$$

Since the integral is an increasing function of c, we can easily find the solution c of this equation by bisection (see, e.g., [3]).

Combining both ideas. If we take both ideas into account, then we need to consider only those roots of the above cubic equation which are larger than or equal to ρ_0; if all the roots are $< \rho_0$, we take $\rho_s(x) = \rho$.

The resulting solution $\rho_s(x)$ depends on the choice of the Lagrange multiplier λ, i.e., in effect, we have $\rho_s(x) = \rho_s(x, \lambda)$. The corresponding value λ can also be similarly determined from the equation $\int \rho_s(x, \lambda)\, dx = D$.

1.4 PREDICTING CLOUD GROWTH: FORMULATION OF THE PROBLEM AND OUR APPROACH TO SOLVING THIS PROBLEM

Why it is important to predict the cloud growth. In the previous sections, when selecting the optimal placement of servers, we assumed that we know the distribution of users' requests, i.e., we know the density $\rho_r(x)$. In principle, the information about the users' locations and requests can be determined by recording the users' requests to the cloud.

However, cloud computing is a growing enterprize, so when we plan to select the server's location, we need to take into account not only the current users' locations and requests, but also their future requests and locations. In other words, we need to be able to predict the growth of the cloud – both of the cloud in general, and of the part corresponding to each specific user location. In other words, we need to predict, for each location x, how the corresponding request density $\rho_r(x)$ changes with time. This density characterizes the size $s(t)$ of the part of the cloud that serves the population located around x.

In the following text, we will refer to this value $s(t)$ as simply "cloud size" – but what we will mean is the size of the part of the cloud that serves a certain geographic location (e.g., the US as whole, or the Southwest part of the US). Similarly, for brevity, we will refer to the increase in $s(t)$ as simply "cloud growth".

How we can predict the cloud growth. To predict the cloud growth, we can use the observed cloud size $s(t)$ at different past moments of time t. Based on these observed values, we need to predict how the rate $\dfrac{ds}{dt}$ with which the size changes depends on the actual size, i.e., to come up with a dependence $\dfrac{ds}{dt} = f(s)$ for an appropriate function $f(s)$, and then use the resulting differential equation to predict the cloud size at future moments of time.

Why this prediction is difficult: the problem of uncertainty. The use of differential equations to predict the future behavior of a system is a usual thing in physics: this is how Newton's equations work, this is how many other physical equations work. However, in physics, we usually have a good understanding of the underlying processes, an understanding that allows us to write down reasonable differential equations – so that often all that remains to be done is to find the parameters of these equations based on the observations. In contrast, we do not have a good understanding of factors leading to the cloud growth. Because of this uncertainty, we do not have a good understanding of which functions should be used to predict the cloud growth.

Main idea of our solution: uncertainty itself can help. The proposed solution to this problem is based on the very uncertainty that is the source of the problem.

Specifically, we take into account that the numerical value of each quantity – in particular, the cloud size – depends on the selection of the measuring unit. If, to measure the cloud size, we select a unit which is λ times smaller, then instead of the original numerical value s we get a new numerical value $s' = \lambda \cdot s$. The choice of a measuring unit is rather arbitrary. We do not have any information that would enable us to select one measuring unit and not the other one. Thus, it makes sense to require that the dependence $f(s)$ look the same no matter what measuring unit we choose.

1.5 PREDICTING CLOUD GROWTH: FIRST APPROXIMATION

How to formalize this idea: first approximation. How can we formalize the above requirement? The fact that the dependence has the same form irrespective of the measuring unit means that when we use the new units, the growth rate takes the form $f(s') = f(\lambda \cdot s)$. Thus, in the new units, we get a differential equation $\dfrac{ds'}{dt} = f(s')$. Substituting $s' = \lambda \cdot s$ into both sides of this equation, we get $\lambda \cdot \dfrac{ds}{dt} = f(\lambda \cdot s)$. We know that $\dfrac{ds}{dt} = f(s)$, so we get $f(\lambda \cdot s) = \lambda \cdot f(s)$.

Solution to the corresponding problem. From the above equation, for $s = 1$, we conclude that $f(\lambda) = \lambda \cdot \text{const}$, i.e., that $f(s) = c \cdot s$ for some constant c.

As a result, we get a differential equation $\dfrac{ds}{dt} = f(s) = c \cdot s$. If we move all the terms containing the unknown function s to one side of this equation and all the other terms to the other side, we conclude that $\dfrac{ds}{s} = c \cdot dt$. Integrating both sides of this equation, we get $\ln(s) = c \cdot t + A$ for some integration constant A. Exponentiating both sides, we get a formula $s(t) = a \cdot \exp(c \cdot t)$ (with $a = \exp(A)$) that describes exponential growth.

Limitations of the first approximation model. Exponential growth is a good description for a certain growth stage, but in practice, the exponential function grows too fast to be a realistic description on all the growth stages. It is therefore necessary to select more accurate models.

1.6 PREDICTING CLOUD GROWTH: SECOND APPROXIMATION

Second approximation: main idea. While it makes sense to assume that the equations remain the same if we change the measuring unit for cloud size, this does not mean that other related units do not have to change accordingly if we change the cloud size unit. In particular, it is possible that if we change a unit for measuring the cloud size, then to get the same differential equation, we need to select a different unit of time, a unit in which the numerical value of time takes the new form $t' = \mu \cdot t$ for some value μ which, in general, depends on λ. Thus, in the new units, we have a differential equation $\dfrac{ds'}{dt'} = f(s')$. Substituting $s' = \lambda \cdot s$ and $t' = \mu(\lambda) \cdot t$ into both sides of this equation, we get $\dfrac{\lambda}{\mu(\lambda)} \cdot \dfrac{ds}{dt} = f(\lambda \cdot s)$. We know that $\dfrac{ds}{dt} = f(s)$, so we get $f(\lambda \cdot s) = g(\lambda) \cdot f(s)$, where we denoted $g(\lambda) \stackrel{\text{def}}{=} \dfrac{\lambda}{\mu(\lambda)}$.

Solving the corresponding problem. If we first apply the transformation with λ_2 and then with λ_1, we get

$$f(\lambda_1 \cdot \lambda_2 \cdot s) = g(\lambda_1) \cdot f(\lambda_2 \cdot s) = g(\lambda_1) \cdot g(\lambda_2) \cdot f(s).$$

On the other hand, if we apply the above formula directly to $\lambda = \lambda_1 \cdot \lambda_2$, we get

$$f(\lambda_1 \cdot \lambda_2 \cdot s) = g(\lambda_1 \cdot \lambda_2) \cdot f(s).$$

By comparing these two formulas, we conclude that $g(\lambda_1 \cdot \lambda_2) = g(\lambda_1) \cdot g(\lambda_2)$. It is well known that every continuous solution to this functional equation has the form $g(\lambda) = \lambda^q$ for some real number q; see, e.g., [1]. Thus, the equation $f(\lambda \cdot s) = g(\lambda) \cdot f(s)$ takes the form $f(\lambda \cdot s) = \lambda^q \cdot f(s)$

From this equation, for $s = 1$, we conclude that $f(\lambda) = \lambda^q \cdot \text{const}$, i.e., that $f(s) = c \cdot s^q$ for some constants c and q. As a result, we get a differential equation

$\dfrac{ds}{dt} = f(s) = c \cdot s^q$. If we move all the terms containing the unknown function s to one side of this equation and all the other terms to the other side, we conclude that

$$\frac{ds}{s^q} = c \cdot dt.$$

We have already considered the case $q = 1$. For $q \neq 1$, integrating both sides of this equation, we conclude that $s^{1-q} = c \cdot t + A$ for some integration constant A. Thus, we get $s = C \cdot (t + t_0)^b$ for some constants C and b (with $b = 1/(q-1)$). In particular, if we start the time with the moment when there was no cloud, when we had $s(t) = 0$, then this formula takes the simpler form $s(t) = C \cdot t^b$.

This growth model is known as the *power function model*; see, e.g., [7, 13, 16].

Discussion. The power function model is a better description of growth than the exponential model – for example, because it contains an additional parameter that enables us to get a better fit with the observed values $s(t)$. However, as mentioned in [13, 16], this model is relatively rarely used to describe the growth rate, since it is viewed as a empirical model, a model that lacks theoretical foundations – and is, therefore, less reliable: we tend to more trust models which are not only empirically valid but also follow from some reasonable assumptions.

In the above text, we have just provided a theoretical foundation for the power function model – namely, we have shown that this model naturally follows from the reasonable assumption of unit-independence. We therefore hope that with such a theoretical explanation, the empirically successful power function model will be perceived as more reliable – and thus, it will be used more frequently.

Limitations of the power function model. While the power function model provides a reasonable description for the actual growth rate – usually a much more accurate description than the exponential model – this description is still not perfect. For example, in this model, the growth continues indefinitely, while in real life, the growth often slows down and starts asymptotically reaching a certain threshold level.

1.7 PREDICTING CLOUD GROWTH: THIRD APPROXIMATION

Third approximation: main idea. To achieve a more description of the actual growth, we need to have growth models with larger number of parameters that can be adjusted to observations. A reasonable idea is to consider, instead of a single growth function $f(s)$, a linear space of such functions, i.e., to consider functions of the type $f(s) = c_1 \cdot f_1(s) + c_2 \cdot f_2(s) + \ldots + c_n \cdot f_n(s)$, where $f_1(s), \ldots, f_n(s)$ are given functions and $c_1, \ldots, c_n$ are parameters that can be adjusted based on the observations.

Which functions $f_i(s)$ should be choose? Our idea is the same as before: let us use the functions $f_i(s)$ for which the change in the measuring unit does not change the class of the corresponding functions. In other words, if we have a function $f(s)$ from the original class, then, for every λ, the function $f(\lambda \cdot s)$ also belongs to the

same class. Since the functions $f(s)$ are linear combinations of the basic functions $f_i(s)$, it is sufficient to require that this property be satisfied for the functions $f_i(s)$, i.e., that we have

$$f_i(\lambda \cdot s) = c_{i1}(\lambda) \cdot f_1(s) + \ldots + c_{in}(\lambda) \cdot f_n(s)$$

for appropriate values $c_{ij}(\lambda)$ depending on λ.

Solving the corresponding problem. It is reasonable to require that the functions $f_i(s)$ be smooth (differentiable). In this case, for each i, if we select n different values $s_1, \ldots, s_n$, then for n unknowns $c_{i1}(\lambda)$, ..., $c_{in}(\lambda)$, we get a system of n linear equations

$$f_i(\lambda \cdot s_1) = c_{i1}(\lambda) \cdot f_1(s_1) + \ldots + c_{in}(\lambda) \cdot f_n(s_1);$$

$$\ldots$$

$$f_i(\lambda \cdot s_n) = c_{i1}(\lambda) \cdot f_1(s_n) + \ldots + c_{in}(\lambda) \cdot f_n(s_n).$$

By using the Cramer's rule, we can describe the solutions $c_{ij}(\lambda)$ of this system of equations as a differentiable function in terms of $f_i(\lambda \cdot s_j)$ and $f_i(s_j)$. Since the functions f_i are differentiable, we conclude that the functions $c_{ij}(\lambda)$ are differentiable as well. Differentiating both sides of the equation

$$f_i(\lambda \cdot s) = c_{i1}(\lambda) \cdot f_1(s) + \ldots + c_{in}(\lambda) \cdot f_n(s)$$

with respect to λ, we get

$$s \cdot f_i'(\lambda \cdot s) = c_{i1}'(\lambda) \cdot f_1(s) + \ldots + c_{in}'(\lambda) \cdot f_n(s),$$

where g' denotes the derivative of the function g. In particular, for $\lambda = 1$, we get

$$s \cdot \frac{df_i}{ds} = c_{i1}' \cdot f_1(s) + \ldots + c_{in}' \cdot f_n(s),$$

where we denoted $c_{ij} \stackrel{\text{def}}{=} c_{ij}'(1)$. This system of differential equations can be further simplified if we take into account that $\dfrac{ds}{s} = dS$, where $S \stackrel{\text{def}}{=} \ln(s)$. Thus, if we take a new variable $S = \ln(s)$ for which $s = \exp(S)$ and new unknowns $F_i(S) \stackrel{\text{def}}{=} f_i(\exp(S))$, the above equations take a simplified form

$$\frac{dF_i}{dS} = c_{i1}' \cdot F_1(S) + \ldots + c_{in}' \cdot F_n(S).$$

This is a system of linear differential equations with constant coefficients. A general solution of such a system is well known: it is a linear combination of functions of the type $\exp(a \cdot S)$, $S^k \cdot \exp(a \cdot S)$, $\exp(a \cdot S) \cdot \cos(b \cdot S + \varphi)$, and $S^k \cdot \exp(a \cdot S) \cdot \cos(b \cdot S + \varphi)$.

To represent these expressions in terms of s, we need to substitute $S = \ln(s)$ into the above formulas. Here,

$$\exp(a \cdot S) = \exp(a \cdot \ln(s)) = (\exp(\ln(s))^a = s^a.$$

Thus, we conclude that the basic functions $f_i(s)$ have the form s^a, $s^a \cdot (\ln(s))^k$, $s^a \cdot \cos(b \cdot \ln(s) + \varphi)$, and $s^a \cdot \cos(b \cdot \ln(s) + \varphi) \cdot (\ln(s))^k$.

Discussion. Models corresponding to $f_i(s) = s^{a_i}$ have indeed been used to describe the growth; see, e.g., [13, 16]. In particular, if we require that the functions $f_i(s)$ be not only differentiable, but also analytical, we then conclude that the only remaining functions are monomials $f_i(s) = s^i$. In particular, if we restrict ourselves to monomials of second order, we thus get growth functions $f(s) = c_0 + c_1 \cdot s + c_2 \cdot s^2$. Such a growth model is known as the *Bass model* [2, 13, 16]. This model describes both the almost-exponential initial growth stage and the following saturation stage.

Oscillatory terms $s^a \cdot \cos(b \cdot \ln(s) + \varphi)$, and $s^a \cdot \cos(b \cdot \ln(s) + \varphi) \cdot (\ln(s))^k$ can be then used to describe the fact that in practice, growth is not always persistent, periods of faster growth can be followed by periods of slower growth and vice versa.

In the above description, we assumed that at each moment of time t and for each location x, the state of the part of the cloud that serves requests from this location can be described by a single parameter – its size $s(t)$. In practice, we may need several related parameters $s^{(1)}(t), \ldots, s^{(k)}(t)$, to describe the size of the cloud: the number of nodes, the number of users, the amount of data processing, etc. Similar models can be used to describe the growth of two or more dependent growth parameters

$$\frac{ds^{(i)}(t)}{dt} = f^{(i)}(s^{(1)}(t), \ldots, s^{(k)}(t)).$$

For example, in the analytical case, the rate of change of each of these parameters is a quadratic function of the current values of these parameters:

$$\frac{ds^{(i)}(t)}{dt} = a^{(i)} + \sum_{j=1}^{k} a_j^{(i)} \cdot s^{(j)}(t) + \sum_{j=1}^{k} \sum_{\ell=1}^{k} a_{j\ell}^{(i)} \cdot s^{(j)}(t) \cdot s^{(\ell)}(t).$$

For $k = 2$, such a model was proposed by Givon et al. [5, 16].

Similar models can be used to describe *expenses* related to cloud computing; see Appendix.

1.8 CONCLUSIONS AND FUTURE WORK

Conclusions. This chapter presents the mathematical solutions for two related cloud computing issues: server placement and cloud growth prediction. For each of these two problems, we first list the simplifying assumptions, then give the derivation of the corresponding model and the solutions. Then, we relax the assumptions and give the solution to the resulting more realistic models.

Future work. The server placement problem is very similar to the type of problems faced by Akamai and other companies that do web acceleration via caching; we therefore hope that our solution can be of help in web acceleration as well.

Acknowledgments

This work was supported in part by the National Center for Border Security and Immigration, by the National Science Foundation grants HRD-0734825 and DUE-

0926721, and by Grant 1 T36 GM078000-01 from the National Institutes of Health. The author is thankful to the anonymous referees for useful advice.

Appendix: Describing Expenses Related to Cloud Computing

Analysis of the problem. The paper [11] analyzes how the price per core C_{core} depends on the per-core throughput T_{core} and on the number of cores N_{core}.

Some expenses are needed simply to maintain the system, when no computations are performed and $T_{core} = 0$. In other words, in general, $C_{core}(0) \neq 0$. It is therefore desirable to describe the additional expenses $\Delta C_{core}(T_{core}) \stackrel{\text{def}}{=} C_{core}(T_{core}) - C_{core}(0)$ caused by computations as a function of these computations intensity. Thus, we would like to find a function $f(s)$ for which $\Delta C_{core} \approx f(T_{core})$.

Main idea. Similar to the growth case, we can use the uncertainty to require that the shape of this dependence $f(s)$ does not depend on the choice of a unit for measuring the throughput – provided that we correspondingly change the unit for measuring expenses.

Resulting formula. As a result, we get a power law $\Delta C_{core} \approx c_T \cdot (T_{core})^b$, i.e., in other words, $C_{core} \approx a + c_T \cdot (T_{core})^b$, where we denoted $a \stackrel{\text{def}}{=} C_{core}(0)$.

Discussion. Empirically, the above formula turned out to be the best approximation for the observed expenses [11]. Our analysis provides a theoretical justification for this empirical success.

Dependence on the number of cores in a multi-core computer. A similar formula $C_{core} \approx a + c_N \cdot (N_{core})^d$ can be derived for describing how the cost per-core depends on the number of cores N_{core}. This dependence is also empirically the best [11]. Thus, our uncertainty-based analysis provides a justification for this empirical dependence as well.

REFERENCES

1. J. Aczel, *Lectures on Functional Differential Equations and their Applications*, Dover, New York, 2006.
2. F. M. Bass, "A new product growth for model consumer durables", *Management Science*, 2004, Vol. 50, Suppl. 12, pp. 1825–1832.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2009.
4. A. Gates, V. Kreinovich, L. Longpré, P. Pinheiro da Silva, and G. R. Keller, "Towards secure cyberinfrastructure for sharing border information", In: *Proceedings of the Lineae Terrarum: International Border Conference*, El Paso, Las Cruces, and Cd. Juarez, March 27–30, 2006.
5. M. Givon, V. Mahajan, and E. Muller, "Software piracy: estimation of lost sales and the impact of software diffusion", *Journal of Marketing*, 1995, Vol. 59, No. 1, pp. 29–37.
6. G. R. Keller, T. G. Hildenbrand, R. Kucks, M. Webring, A. Briesacher, K. Rujawitz, A. M. Hittleman, D. J. Roman, D. Winester, R. Aldouri, J. Seeley, J. Rasillo, T. Torres, W. J. Hinze, A. Gates, V. Kreinovich, and L. Salayandia, "A community effort to construct a gravity database for the United States and an associated Web portal", In: A. K. Sinha (ed), *Geoinformatics: Data to Knowledge*, Geological Society of America Publ., Boulder, Colorado, 2006, pp. 21–34.
7. G. Kenny, "Estimating defects in commerical software during operational use", *IEEE Transactions on Reliability*, 1993, Vol. 42, No. 1.

Scalable Computing: Theory and Practice, 1st edition. By Samee U. Khan, Lizhe Wang, and Albert Y. Zomaya
Copyright © 2012 John Wiley & Sons, Inc.

8. R. Kershner, “The number of circles covering a set”, *American Journal of Mathematics*, 1939, Vol. 61, No. 3, pp. 665–671.

9. C. Kiekintveld and O. Lerma, “Towards optimal placement of bio-weapon detectors", *Proceedings of the 30th Annual Conference of the North American Fuzzy Information Processing Society NAFIPS’2011*, El Paso, Texas, March 18–20, 2011.

10. O. Lerma, E. Gutierrez, C. Kiekintveld, and V. Kreinovich, “Towards Optimal Knowledge Processing: From Centralization Through Cyberinsfrastructure to Cloud Computing", *International Journal of Innovative Management, Information & Production (IJIMIP)*, 2011, Vol. 2, No. 2, pp. 67–72.

11. H. Li and D. Scheibli, “On cost modeling for hosetd enterprise applications”, In: D. R. Avresky (ed.), *Cloudcomp 2009*, Lecture Notes of the Institute of Computer Sciences, Social-Informatics, and Telecommunications Engineering, Springer Verlag, 2010, Vol. 34, pp. 261–269.

12. L. Longpré and V. Kreinovich, “How to Efficiently Process Uncertainty within a Cyberinfrastructure without Sacrificing Privacy and Confidentiality", In: N. Nedjah, A. Abraham, and L. de Macedo Mourelle (Eds.), *Computational Intelligence in Information Assurance and Security*, Springer-Verlag, 2007, pp. 155–173.

13. H. Pham, *Handbook of Engineering Statistics*, Springer Verlag, London, 2006.

14. P. Pinheiro da Silva, A, Velasco, M. Ceberio, C. Servin, M. G. Averill, N. Del Rio, L. Longpré, and V. Kreinovich, “Propagation and Provenance of Probabilistic and Interval Uncertainty in Cyberinfrastructure-Related Data Processing and Data Fusion", In: R. L. Muhanna and R. L. Mullen (eds.), *Proceedings of the International Workshop on Reliable Engineering Computing REC’08*, Savannah, Georgia, February 20–22, 2008, pp. 199–234.

15. A. K. Sinha (ed), *Geoinformatics: Data to Knowledge*, Geological Society of America Publ., Boulder, Colorado, 2006.

16. G. Zhao, J. Liu, Y. Tang, W. Sun, F. Zhang, X. Ye, and N. Tang, “Cloud computing: a statistics aspect of users”, In: M. G. Jattun, G. Zhao, and C. Rong (eds.), *CloudCom’2009*, Springer Lecture Notes in Computer Science, 2009, Vol. 5931, pp. 347–358.