# Towards Elastic Algorithms as a New Model of Computation for the Cloud

YIKE GUO, MOUSTAFA M. GHANEM, RUI HAN
Department of Computing, Imperial College London, London, UK

Cloud computing has emerged as a cost-effective way for delivering metered computing resources. It supports a Pay-as-you-go model of computation in which users pay only for the resources used, when used. Within this context managing resource elasticity has become an active research topic for the research community. A major focus of such research has been on investigating various approaches to support dynamic scaling of the resources used so as to match the users computational demands while minimising the cost of using such resources. However, little or no work has investigated the concept of supporting algorithm elasticity itself, i.e. organizing the users computation to adapt dynamically to resource availability and cost. In this paper we introduce the concept of an elastic algorithm (EA), and algorithm that structures the computation to make use of the Pay-as-you-go paradigm. In contrast to conventional algorithms, where a computation is typically regarded as a deterministic process that only produces an all-or-nothing result, an EA is organized to generate a sequence of approximate results corresponding to its resource consumption. On a tight resource budget the algorithm guarantees producing an approximate useful result. However, if the user has more budget, and accordingly can use more resources, an EA guarantees producing better results. In this sense, the quality of the algorithm output becomes elastic to its resource consumption. In this paper, we formalize the properties of algorithm elasticity and also formalize the key features of an EA. We illustrate the key concepts by designing an elastic kNN classification algorithm and discussing its key elastic features. We also describe a number of key challenges that need to be addressed when designing EAs in general and set them as a research agenda for the community.

Keywords: Cloud computing, Pay-as-you-go, elastic algorithms

## 1. INTRODUCTION

Cloud computing has emerged as a cost-effective paradigm for delivering metered computing resources. Within the Cloud paradigm, hardware and/or software computational resources are provided as a utility that is shared between multiple users. Each user is provided a piece of solely owned virtual resource instance. Moreover, based on a *Pay-as-you-go* business model, users are allowed to acquire and release resources on demand and are billed only for the resources they use. This feature, of being able to scale-up and down resources used on demand, is typically described as resource elasticity.

Elasticity management coupled with the new *Pay-as-you-go* cloud business models give rise to various new challenges that require revisiting our assumptions about how we design programs and algorithms. To date, most research [Hwang et al. 2012; Moreno-Vozmediano et al. 2009; Sharma et al. 2011] on supporting elasticity management in Cloud environments has focused on either the provision of mechanisms that simplify the dynamic acquisition/release of resources based on the variation of the users computational demand or on developing capacity planning and scheduling algorithms that help in minimizing the execution costs of executing programs in a Cloud environment. These methods allow us to address easily questions of the type: *How much would it cost me to get my result by a particular time?*, or *How can I schedule the use of resources to minimize the costs of generating my results?*.

However, little or no work has been conducted on investigating the concept of algorithmic elasticity, i.e., how can we can adapt the output results of a computation to the amount and price of available resources. In particular, there is no easy way to answer questions such as *Given fixed budget, what quality of results can we obtain from our program?*; or *Given a fixed budget and deadline constraints, how can we adapt the quality of our algorithms outputs?*; or *How should the quality of our algorithms results vary with resource price fluctuations?*. We may even feel that such all questions are ill-formed for two reasons. Our view of a traditional algorithm is that it

has a one-off answer only; either it produces a result or it fails to produce a result - there is no elasticity within such algorithms at all, they are not designed to adapt to resource price variations and traditionally have no notion of result quality. This view limits us in exploring many of the opportunities and benefits that a *Pay-as-you-go* business model of computation can offer.

This paper is an extended and completely revised version of a preliminary version presented in IEEE CloudCom 2012 [Guo et al. 2012]. Here, we describe the development of a class of elastic algorithms that are well-suited for the *Pay-as-you-go* computing model, allowing us to pay by quality of result. In contrast to conventional algorithms, where computation is a deterministic process with an all-or-nothing result, an elastic algorithm (EA) is designed to generate a sequence of approximate results whose quality is proportional to its resource consumption. We view such algorithms to be elastic in nature since on a tight budget the user is guaranteed a usable result, however, if more budget is available, the algorithm guarantees better quality results and vice versa. Our aim in this paper is to explore the desirable properties of EAs in a Cloud environment and also to define, formally, what we mean by elasticity at the algorithm, rather than resource, level.

The paper is organized as follows. In Section 2, we present background to resource elasticity management and elaborate on our motivations for investigating algorithmic elasticity. We also review some existing methodologies for designing adaptive algorithms traditionally used in real-time systems that can provide insights on how to design elastic algorithms for Cloud computing. In Section 3, we present the basic concepts and definitions for our EA methodology. In particular, we present a formal definition for a class of elastic algorithms and their key properties. We also present a formal definition of algorithmic elasticity based on the economic definition of elasticity. In Section 4, we describe a generic methodology for designing elastic algorithms over large data sets based on using hierarchical data structures. We also provide a detailed example for designing an elastic k-nearest neighbor (kNN) classification algorithm to illustrate the key features of our approach. In Section 5, we present our conclusions and describe an ambitious research agenda for investigating the properties of elastic algorithms.

## 2. MOTIVATION AND RELATED WORK

### 2.1 Resource Elasticity

For years, we have had a simple view of an algorithm: it is a sequence of computational steps that should produce a deterministic result after consuming some resource. Since the algorithms output result is prescribed, its computational properties are typically measured by its computational (time and space) complexity as the problem size grows. Similarly, the properties of its implementation are measured by performance metrics, such as response time or throughput. This view forms the basis of the modern concept of quality of service (QoS) where software is provided as a service, and where users pay for the resources used to satisfy their QoS requirement.

A typical example of applying this traditional view in Cloud environments can be seen in managing multi-tier web applications [Han et al. 2012], such as e-commerce sites or other applications that service multiple users. The QoS requirements for the implementation are typically expressed as response time and/or throughput, effectively measuring the performance of producing a result for each request. When the demand for application increases (measured by the number of requests submitted by end users), the application provider is traditionally willing to pay more so as to maintain the performance of the application as seen by its users. When the demand decreases, the application owner is not willing to pay for idle resources. Elasticity management in this case enables real-time acquisition/release of computing resources used, at each tier of the application, either up and down so as to meet the QoS requirements while minimizing the monetary costs paid for the resources used [Han et al. 2011] [Han, Ghanem, Guo, Guo and Osmond 2012] [Han et al. 2012]. The resource usage, in this case, can be described as being elastic with respect to the user demand, and also with respect to the price of resources. We note that exploiting elasticity here does not change the output of the program as each user still receives the exact prescribed

result of the computation. Rather, it is used mainly to change the performance characteristics of the computation.

We note that elasticity management becomes slightly more complicated, albeit still manageable, if the price of resources used varies over time. Various Cloud providers, such Amazon AWS , provide resource pricing schemes where prices vary dynamically according to supply and demand conditions. For example, under a spot price scheme, users are allowed to bid for computational resources and gain access to them so long as the providers offer price is lower than the users bid price. If the offer price becomes higher than the bid price, the cloud provider reserves the right to terminate the users computation. A key implication of such model is that users have to take into consideration such price fluctuations when making their resource provisioning decisions. They now not only have to minimize the costs of executing their computation but also have to ensure that the deadlines for obtaining the computations results are also met. Another practical implication is that the implementation of the algorithm now needs to incorporate check points where the execution of the algorithm can be suspended and then safely resumed later. It even becomes desirable that some kind of meaningful partial results or useful results can be returned at such check points to ensure that the investments already made towards the computation are not lost if the user cannot resume the program later.

### 2.2 Algorithm Elasticity

In this paper, we depart from the resource-oriented of elasticity view with a clear objective in mind; investigating the concept of elasticity at the algorithm level, rather than at the resource level. We are motivated by the question of whether money can buy something else rather than just resources to improve performance? How about if we consider another form of elasticity: the elasticity of the algorithms outputs with respect to resource used? In this case, we may be willing to pay more (use more resources) to obtain better quality of results, not simply better performance. The challenge now becomes how to organize our computations to exploit such result quality elasticity.

To address our objective we investigate algorithms that generate a sequence of improving approximate results whose result quality, based on same measure, is proportional to their resource consumption. As more resource is consumed, better results will be derived. We can illustrate the concept using an image rendering application as an example. A conventional rendering algorithm is traditionally designed to generate the final result with the highest resolution. In a Cloud environment the user has no option but to pay the high cost for using the resources required to produce the best result. However, on a limited budget, it may make more sense to adopt an incremental rendering method; i.e. the algorithm could start by producing an approximate, but acceptable, result using a limited amount of resources (or budget) and return this to the user. If the user has more budget then the algorithm can continue to refine the image to improve its resolution by using more resources. In this case the quality of the result could be regarded as being elastic with respect to the resource usage, and we can easily call an algorithm with such behavior an elastic algorithm (EA). It is not difficult to see that similar EAs that trade-off result quality with resource usage can be designed and used in a wide range of domains, including numerical, scientific and engineering computations, statistical estimation and prediction in data mining applications, heuristic search applications and database query processing applications where generating approximate cheaper answers may be acceptable to the user.

### 2.3 Related Methodologies

Our proposed concept of EAs builds on lessons learnt from previous methodologies used outside the Cloud computing area, and especially those designed for developing adaptive, or flexible, algorithms in the context of real-time applications executing on environments with limited resources. We can summarize the key methodologies traditionally used in such applications into two camps as described below.

**Resource-aware Algorithms**: The resource-aware algorithm methodologies, e.g., [Gaber

and Yu 2006; Poladian et al. 2004], focus on organizing computation to produce the best possible results on devices with limited resources, e.g. mobile devices or sensor nodes. On such devices resources such as memory, processing cycles, communication bandwidth and battery life may degrade, or vary, with time as time. The methodology controls how an algorithm adapts its use of resource dynamically in response to such changes. In [Gaber and Yu 2006], three key control strategies are proposed, these are: 1) Controlling the algorithms Input Granularity, e.g. by changing the resolution or details of the input data structures; 2) Controlling the algorithms Processing Granularity, e.g. by performing less or more computation and 3) Controlling the algorithms Output Granularity, e.g. by controlling the resolution or detail of the output data structures.

The resource-aware algorithms approach typically requires the implementation of a real-time resource monitor and a decision mechanism (e.g. set of rules) for choosing between different implementations of individual steps in the algorithms implementation. The reactive approach builds implicitly on knowledge of how the quality of results varies with resources but does not necessarily require the definition and use of an explicit quality function or metric.

**Anytime Algorithms**: Anytime algorithms are used in real-time applications and adopt the mechanism of expectation-driven iterative refinement promoted by Boddy and Dean [Dean and Boddy 1988; Dean 1989] to return approximating answers from a particular algorithm at any point in time. The algorithm itself is thus designed to return approximate answers that can be improved when given more time, e.g. by using an incremental result refinement strategy. Zilberstein [Zilberstein 1996] defined the key features typically exhibited by anytime algorithms as: (1) Interruptibility and Pre-emtability: the anytime algorithm can be interrupted at any time and return an approximate solution. Moreover, the algorithm can be re-started at a later time to continue its computation. (2) Explicit Measurable and Recognisable Quality: a precise and computable quality function is available to enable the determination of the quality of an approximating solution during the run-time of the algorithm. (3) Monotonicity, Consistency and Diminishing Returns of Quality: The quality function has an increasing behaviour as time increases. Moreover, a correlation needs to exist between the input quality, computation time and resulting quality to enable predictions for the resulting quality. Due to the approximating nature of the algorithms their rate of improvement is generally reduced as the computation progresses and as the algorithm converges on the final highest quality result. Zilberstein [Zilberstein 1996] described how using various dynamically measured quality metrics, such as accuracy (difference of the current approximate solution to the exact optimal solution) and certainty (the confidence level of correctness), can be used to guide the progress of the approximation process of a variety of algorithms.

The standard anytime algorithm approach itself does not monitor resource usage in real-time and assumes that time is the only key resource of concern. It also does not attempt to optimize the use of resources and traditionally keeps an algorithm running until either a time deadline is reached or the final result produced. However, other research has investigated scheduling of resources between multiple approximation-based programs so as to maximize the overall utility of the computations. Examples include work on flexible computing by Horvitz [Horvitz 1988; Horvitz 1990] in the context of decision support applications and work on scheduling imprecise numerical computations by Liu et al [Liu et al. 1991]. The aim of all such pre-Cloud work was producing the best possible set of results while meeting real-time deadline constraints.

## 3. TOWARDS ELASTIC ALGORITHMS

Both the resource-aware algorithm methodology and the anytime algorithm methodology provide valuable insights on how we could develop elastic algorithms for Cloud environments. The anytime algorithm methodology provides a generic approach for designing algorithms that incrementally improve their output results along a computing process. In contrast, the resource-aware algorithms methodology provides the notion of trade-off between the quality of computation

results and the available resources. Combining both schools, it is possible to investigate the development of the new paradigm of EAs for Cloud environments where the quality elasticity of the computation results with respect to resources used. We also note that neither methodology explicitly formalized the notion of elasticity itself at the computation level. Moreover, neither provides a framework for developing and reasoning about the elasticity properties of the algorithm. Based on these observations, this section formalizes our concept of elastic algorithms for Cloud computing and also formalize the properties of algorithm elasticity.

### 3.1 Definition: Elastic Algorithms

We define a class of elastic algorithms where EA is an algorithm that generates a sequence of approximate results, with each result being associated with a quality measurement that is proportional to resource used to produce it. With an EA, an algorithm $A$ produces an approximate result, by spending, $S_{i+1}$ by spending a specific investment $\triangle I$ towards resource usage for refining a previously generated approximate result $S_i$. The computation can proceed, incrementally, to produce results with better quality if more investment budget is available. We define this formally as follows:

**Definition 1 (elastic algorithm)**: *An EA,* A*, takes an investment* $\triangle$I *and the existing best result* $S_i$ *at step i to generate an improved result:* $S_{i+1}$=A($\triangle I$, $S_i$) *such that the following four features hold:*

**Measurable quality**: *For any result S, there is a computable quality function:* $Q(S) >= 0$.

**Meaningful results**: *there is a quality measurement* $\varepsilon$ *such that* $Q(S_i) = \varepsilon$ *and for any i>1,* $Q(S_i) >= Q\varepsilon$*, where* $S_1$ *is the first result generated by A.*

**Quality monotonicity**: *For any i>1,*

$$Q(S_{i+1}) >= Q(S_i) \tag{1}$$

**Accumulative computation**:

$$S_k = A(\triangle I_{ij}, S_j) = A(\triangle I_{ij} + \triangle I_{jk}, S_i), \tag{2}$$

*Where* $\triangle I_{ij}$, $\triangle I_{jk}$, *and* $\triangle I_{ik}$ *are the investments needed to convert* $S_i$ *to* $S_j$, $S_j$ *to* $S_k$ *and* $S_i$ *to* $S_k$*, respectively for any 0<=i<j<k*

The first property, measurable quality, means that an explicitly defined and measurable quality function can be computed for each approximate result. The second and third properties, measurable quality, and quality monotonicity mean that each approximate result must be a complete, rather than partial, output from the computation so that it is useful to the user. They also indicate that there is a minimum acceptable quality threshold associated with the first produced result and that quality improves monotonically as more results are refined. The fourth property, accumulative computation, means that a particular result $S_k$ can be derived by refining previous results $S_i$ or $S_j$ in either single investment in computation or multiple investments in computation.

We can illustrate the meaning of the above properties by considering an incremental image rendering algorithms [Pharr and Humphreys 2004].

*Measurable quality*: in incremental image rendering, the quality of the result can be quantitatively measured by the resolution (or samples-per-pixel) of the generated image (e.g., 100×100 pixels).

*Meaningful results*: the overall process of an incremental image rendering algorithm can produce a series of approximate results, which are full rendered images. For any approximate result Q($S_i$) where i>1, we have Q($S_i$)>=1×1 pixels.

*Accumulative computation*: in incremental image rendering algorithms, a resumable image file can be used as a starting point to resume the rendering process without calculating the samples that have already been processed. Suppose $S_i$ is the result in $investment_i$ with resolution $pix_i \times pix_i$. Starting from the resumable file of $S_i$, the algorithm can get result $S_{i+1}$ using I. In contrast, starting from any previous resumable file of result $S_j$ where 0<=j<i, the algorithm

needs I+$\triangle I_{ij}$ to get result $S_{i+1}$ and $\triangle I_{ij}$ represents the investment needed to render the image from $pix_j \times pix_j$ to $pix_i \times pix_i$.

*Monotonicity of quality*: in incremental image rendering, each new investment can guarantee to generate an image with a higher resolution/quality, e.g., from 100×100 pixels to 1000×1000 pixels.

### 3.2 Algorithmic Elasticity

We note that elasticity has a precise meaning as economic term [Samuelson. and Nordhaus. 2011]. It is the measurement of how changing one economic variable affects others. The elasticity of y with respect to x, $E_y^x$ , is defined as $E_y^x = \frac{\frac{\partial y}{\partial x}}{\frac{x}{y}}$.

Given this definition, we can characterizes any *Pay-as-you-go* Cloud computing business model by the price elasticity with respect to demand, and we can also characterize any property of a Cloud application by its elasticity with respect to resource or cost. This would allow us to develop a methodology for developing Cloud applications that are aware of how the price of resources varies along the computation and that would enable us to reason about the various trade-offs that exist when executing a program. Developing such methodology would ideally incorporate explicitly the concepts of both quality elasticity $E_Q^R$ and price elasticity $E_Q^P$. These metrics would then enable us to investigate how the quality of an algorithms output is affected by the available budget and resource prices.

The quality elasticity with respect to the investment $E_Q^I$ characterises the key property of an EA. In a Cloud model, we can express this elasticity in different ways. Let $I$=P×Q be the cumulative investment required to produce an improved result starting from the existing result $S$. We can define the monotonicity of quality in definition 1 in an equivalent way:

$$Q(A(\triangle I, A(I, S))) >= Q(A(I, S)) \tag{3}$$

Interrupting and resuming the EA would incur extra cost:

$$Q(A(I + \triangle I, S))) >= Q(A(\triangle I, A(I, S))) \tag{4}$$

Thus, given an EA $A$, the quality $Q$ is a function of the cumulative investment $I$ and a starting result $State$: $Q = q(I, State)$. Using the quality function, we can define the investment elasticity $E_Q^I$:

**Investment elasticity**: the elasticity of quality with respect to the investment:

$$E_I^Q = \frac{\partial Q}{\partial I} \times \frac{I}{Q} \tag{5}$$

In a Cloud model, an investment is modelled by the consumed resources as well as by the price of these resources. We define functions $I = i(R, P)$ and $S = s(R, P)$ where $R$ is the resource and $P$ is the price (i.e., per unit of resources). We then further define the resource and price elasticity.

**Resource elasticity**: the elasticity of quality with respect to the resources used:

$$E_R^Q = (\frac{\partial Q}{\partial I} \times \frac{\partial I}{\partial R} + \frac{\partial Q}{\partial S} \times \frac{\partial S}{\partial R}) \times \frac{R}{Q} \tag{6}$$

**Price elasticity**: the elasticity of quality with respect to the price:

$$E_P^Q = (\frac{\partial Q}{\partial I} \times \frac{\partial I}{\partial P} + \frac{\partial Q}{\partial S} \times \frac{\partial S}{\partial P}) \times \frac{P}{Q} \tag{7}$$

## 4. AN EXAMPLE OF DEVELOPING AN EA

Developing an EA is not necessarily a trivial task and converting a traditional algorithm to an elastic one is always easy. In practice, many algorithms do not necessary have a natural algorithmic structure that supports the approximation of results. Moreover, there is not always

an obvious quality function that can be used to assess the quality of the results for each successive result, and that also ensures that the quality of the results improve (or at least not deteriorate) as more investment is used. In this section, we describe a general process of developing an EA operating on large data structures. We then provide an example of developing an example EA for the popular kNN classification algorithm used in machine learning applications. Finally, we discuss how the behaviour of our algorithm satisfies the four desired elastic features in a cloud environment.

The pseudocode in Figure 1 describes the general skeleton of an EA operating on a large data structure. As pre-processing, or construction step, the EA first transforms, or codes, the original input data structure using a hierarchical representation that summarizes the data at successive levels of granularity (line 2), where each granularity level can be used to produce an approximate result. One data structure that can be used for this purpose is the R-tree [Guttman 1984] which indexes a large collection of data points and where nodes at different depths of the R-tree represent data points at different levels of granularities. Subsequently, the algorithm enables users to make an investment to produce the first approximate result from the initial state based on the coarsest granularity representation of the data. Starting from the obtained result, users can make a further investment to generate improved approximations of the result by using codes with finer granularities.

**EA Skeleton**

**Input:** Input data.

**Output:** A list of successive approximate results.

1. **Begin**
2. Apply a hierarchical coding method to transform the input data into codes with different levels of granularities; // *Construction step*
3. **If** users have an investment **then**
4. Perform an elastic step to produce the first approximate result using a code with a coarse level of granularity; // *Initialisation step*
5. **While** user has an extra investment
7. Perform an elastic step using a code with a finer level of granularity to produce next refined result; // *Refinement step*
8. **End**

Figure. 1: The EA skeleton.

### 4.1 The R-tree Coding Method to Support the Elastic kNN Algorithm

The kNN algorithm is a popular classification algorithm used in many fields. Given a test point *x'*, the nave kNN algorithm linearly scans all the training points to find the k nearest neighbours of point *x'*. Point *x'* is then classified by a majority vote of its $k$ nearest neighbours, i.e. *x'* is assigned to the class most common amongst its $k$ nearest neighbours [Webb et al. 2011].

The basic concept behind applying the R-tree to transform the nave kNN algorithm into an elastic version is to convert the traditional linear search of training points into a hierarchical search of abstracted training samples with different levels of granularities. This hierarchical index structure allows users choose any granularity of abstract samples to perform classification. At a coarser granularity, there are less abstract samples and each abstract sample represents more original training samples. Thus, classifying such abstract samples requires less computation, but also produces results that are less accurate. Users can also choose a finer granularity, thus

obtaining more precise information of these samples and making more expensive and accurate predictions.

An R-tree node has one or multiple entries. An entry of a leaf node represents one d-dimensional training point x. An entry of a non-leaf node refers to one of its child nodes. Figure 2 shows an example of two R-tree for indexing two-dimensional training points. At a lower depth of the tree (e.g., the root node at depth 0), there are less nodes and each node corresponds to more training points, thus representing the data at a coarser granularity. For example, at depth 0 of Figure 2s R-tree, there is only one root node for each R-tree. In contrast, at depth 2 of each R-tree, there are seven nodes and each node represent training points at a finer granularity.

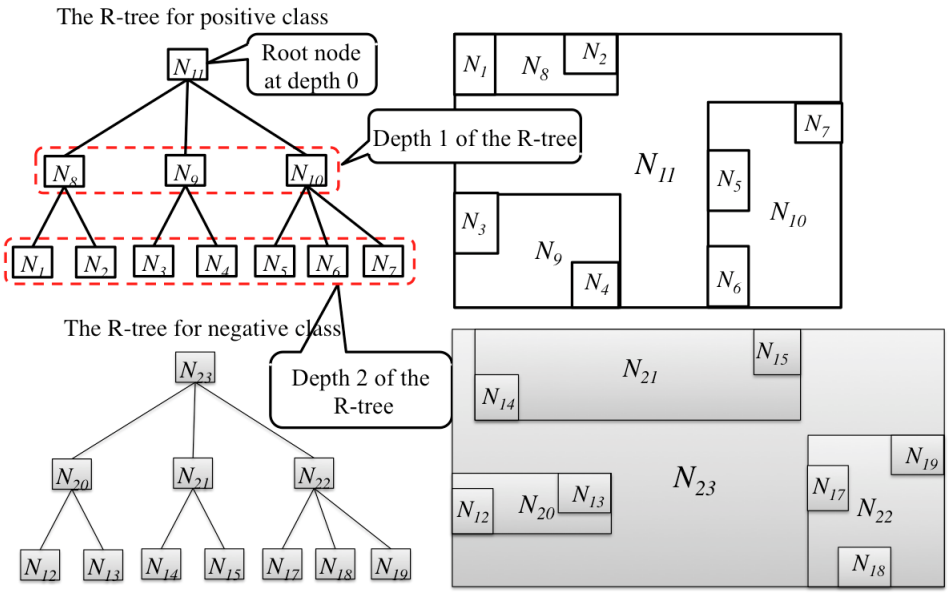



Figure. 2: An example two-dimensional R-trees.

Note that a separate R-tree is applied to index training samples from each class. In other words, there are $p$ R-trees to be generated for a p-class training set. Only the training points (without class labels) are stored in the R-tree nodes, because each R-tree represents one class and each node represents the summarised information of multiple training points from the same class. In Figure 2s example, two R-trees are used to indexes training points from the positive and negative classes, respectively.

## 4.2 The Elastic kNN Algorithm

Following the general process of an elastic algorithm in Section 4.1, the elastic kNN algorithm takes a training set and a test point as input data and generates one initial result and multiple refined results for users as output. The detailed algorithm is given in Figure 3. At the first construction step (line 2), the algorithm generates p (p>1) R-trees using the standard R-tree construction algorithm for the training set with p classes [Guttman 1984]. At the initialisation step (line 4), users have multiple options of approximate results by selecting nodes at different depths of the tree. Depending on users investment, the algorithm can select the nodes at any particular depth of the $p$ R-trees and use them as abstract training samples to produce an approximate result. The root nodes at depth 0 represent training samples at the coarsest granularity and the leaf nodes at the deepest depth denote the finest granularity of training samples. The nave kNN classification algorithm is used in each elastic step and is applied to scan linearly the nodes at one level to find the test points $k$ nearest neighbours. In the search, the distance between the

test point and an R-tree node is calculated as the maximal Euclidean distance between the point and the nodes rectangle. At a refinement step (line 7), once users make an extra investment, the algorithm can select nodes at a deeper depth of the p R-trees as abstract training points. These nodes represent the original training points at a finer granularity, thus producing an approximate result with better quality.

**The elastic kNN algorithm**

**Input:** Training set and a test point.

**Output:** A list of successive approximate values of predicted class for test point

1. **Begin**
2. Apply hierarchical R-tree coding method to transform training set into a list of codes with different levels of granularity; // *Construction step*
3. **If** users have an investment **then**
4. Perform the naïve kNN classification algorithm to produce the first approximate result using a code consisting of nodes at depth 1 of the R-tree; // *Initialisation step*
5. **While** user has an extra investment
7. Perform the naive kNN classification algorithm to produce next refined result using a code consisting of nodes at a deeper depth of R-trees; // *Refinement step*
8. **End**

Figure. 3: The elastic kNN algorithm.

Figure 4 shows an example of elastic 3NN ($k$=3) algorithm on a two-dimensional dataset. At the construction step, two R-trees are generated to index positive and negative training points, respectively. Given a test point $x$ to be classified, at the initialisation step, the six nodes at depth 1 of the two R-trees are used as abstract training points and the nodes $N_9$, $N_11$ and $N_56$ are selected as $x$s nearest neighbours. Thus, $x$s class label is predicted as positive. At a refinement step, the 14 nodes at depth 2 of the R-trees are used as abstract training points. These nodes represent the training points at a finer granularity. The algorithm then updates the classification results: $N_4$, $N_5$ and $N_6$ are selected as $x$s nearest neighbours.

### 4.3 Definition of the Quality Measure for the Elastic kNN Algorithm

We define the quality of elastic kNN algorithm according to the Receiver operating characteristic ($ROC$) paradigm, which has been proved to perform very well in many classification problems [Fawcett 2006]. Specifically, we apply the area under the $ROC$ curve ($AUC$) as the single-number quality metric [Hand and Till 2001] for the elastic kNN algorithm. In the following, we first introduce the generic $AUC$ metric for evaluating classification algorithms, and then explain how to interface this metric to our elastic kNN algorithm. Based on this quality measure, we further discuss the four features and demonstrate elastic behaviors of the elastic kNN algorithm.

In $ROC$ space, given a finite testing set with $n^P$ positive prediction objects and $n^N$ negative objects where $n = n^P + n^N$, two quality metrics are used in $ROC$ to describe the classification result. The first metric is True Positive Rate ($TPR$ or hit rate): $TPR = n_{TP}/n^P$ where $n_{TP}$ is the number of positive testing points that are classified as positive. $TPR$ defines the proportion of correctly classified positive points among all positive data points, e.g. in a diagnostic test, $TPR$ is the probability of detecting a true patient. The second metric is False Positive Rate (FPR): $FPR = n_{FP}/n^N$ where $n_{FP}$ is the number of negative testing points that are classified as positive. $FPR$ defines the proportion of incorrectly classified positive points among all negative testing points. For example, in a medical example, $FRP$ represents the probability of judging

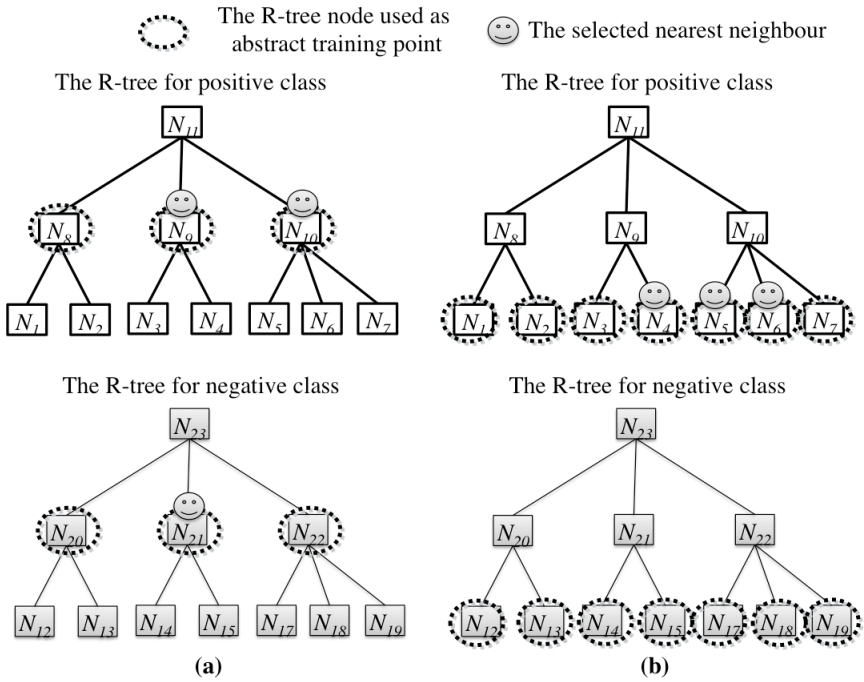



Figure. 4: An example elastic kNN algorithm on 1-dimensional data.

healthy people as diseased. Note that for simplicity, we discuss binary classifier in this work, however, our results can be extended easily to multiple classes.

In a binary classifier, $AUC$ represents the probability that a randomly chosen positive testing point will have a larger estimated probability of belonging to positive class than a randomly chosen negative testing point.

Given a finite test set with $n^p$ positive testing points and $n^N$ negative testing points, where $n = n^P + n^N$, we rank these points in increasing order according their estimated probabilities of belonging to the positive class. In other words, the point with the largest estimated probability of belonging to positive class $c_P$ has the highest ranking order.

Let $x$ be a positive testing point and its ranking order is $i$ in $n_P$ positive points ($1 <= i <= n^P$) and $rank_i$ in all $n$ points ($1 <= rank_i <= n$), there are ($rank_i - i$) negative points whose ranks are lower than x. When considering all $n^P$ positive points, we can summarize the total numbers that positive points have higher ranking orders, namely larger estimated probabilities of belonging to the positive class, than negative points:

$$\sum_{1}^{n^P}(rank_i - i) = \sum_{1}^{n^P} rank_i - \sum_{1}^{n^P} i = \sum_{1}^{n^P} rank_i - n^P \times (n^P + 1)/2 \tag{8}$$

In the ideal situation, each positive point has higher rank than all $n_N$ negative points. Thus, $n^P \times n^N$ represents this ideal situation that each of positive points ranking order is larger than all the $n_N$ negative points. Thus, we can calculate the $AUC$ quality as:

$$AUC = \frac{\sum_{1}^{n^P}(rank_i) - (n^P \times (n^P + 1))/2}{n^P \times n^N} \tag{9}$$

Based on the generic method of calculating $AUC$, we explain how to calculate $AUC$ in the elastic $k$NN algorithm. Given a testing point $x$ and the number of nearest neighbours $k$, $x$'s estimated probability of belonging to positive class, denoted by $p(c_P|x)$, is decided by its number of positive nearest neighbours $k^P$: $p(c_P|x) = \frac{k^P}{k}$. Intuitively, this means the more positive nearest

neighbours of $x$ has (the larger the $k_P$), the larger $x$s estimated probability of belonging to the positive class. We can calculate the $AUC$ metric of the elastic kNN classification algorithm given a testing set with $n^P$ positive testing points and $n^N$ negative testing points:

$$AUC = \frac{\sum_1^{n^P}(rank(\frac{k_i^P}{k})) - (n^P \times (n^P + 1))/2}{n^P \times n^N} \tag{10}$$

Where $x_i$ is a positive testing point, $k_i^P$ is its number of positive nearest neighbours and $k$ is the number of nearest neighbours.

### 4.4 The Four Features of the Elastic kNN Algorithm

Based on the defined quality measure for the elastic kNN algorithm, the algorithm can meet the four desired EA features in Definition 1: 1) *Measurable quality*: the quality $AUC$ is a computable function; 2) *Meaningful results*: the quality of any approximate result has a minimal value: 0.5, which makes the result meaningful; 3) *Accumulative computation*: as the algorithm proceeds it explores more nodes in the trees and return better approximation of the points true class; 4) *Monotonicity of quality*: in the elastic kNN algorithm, positive/negative prediction objects have higher probability to increase their positive/negative nearest neighbours. Thus, the algorithm can improve the quality by increasing the ranks of positive objects, or equivalently decreasing the ranks of negative prediction objects.

### 4.5 Elastic Behaviour of the Elastic kNN Algorithm

We demonstrate elasticity behaviour of the algorithm using simple theoretical examples. Table I shows an example of 10 test points with five positive ones and five negative ones. Using the proposed elastic kNN algorithm, eight approximating results $S_1$ to $S_8$ are produced using eight depths of the R-trees, respectively. In Table I, one line represents one result, and the 10 columns denote the ranking orders of the 10 test points ($RO$ for ranking order), where + denotes a positive test point and - denotes a negative test point.

For each result, each test point $x$s nearest neighbours are explored and the number of nearest neighbours from the positive class is updated. This influences the ranking orders of the five positive testing points, which determine the quality ($AUC$ values) of each result.

Table I: An Example of AUC Values for the Elastic kNN Algorithm

| ***Result*** | ***RO* 10** | ***RO* 9** | ***RO* 8** | ***RO* 7** | ***RO* 6** | ***RO* 5** | ***RO* 4** | ***RO* 3** | ***RO* 2** | ***RO* 1** |
|---|---|---|---|---|---|---|---|---|---|---|
| $S_1$ | + | - | + | - | + | - | - | + | - | + |
| $S_2$ | + | - | + | - | + | - | + | + | - | - |
| $S_3$ | + | - | + | - | + | + | + | - | - | - |
| $S_4$ | + | + | + | - | - | + | - | + | - | - |
| $S_5$ | + | + | + | - | + | - | + | - | - | - |
| $S_6$ | + | + | + | + | - | - | + | - | - | - |
| $S_7$ | + | + | + | + | - | + | - | - | - | - |
| $S_8$ | + | + | + | + | + | - | - | - | - | - |

In the example of Figure 5, we make a simple assumption that a fixed investment $\triangle I = 1$ (dollar) is consumed to produce an improved result $S_{i+1}$ starting from result $S_i$ for any $0 <= i < 7$. Figure 5(a) shows that the quality ($AUC$ value) is a monotonic function of the cumulative investment I. The quality improvement is larger at the early stages of the investment $\triangle I$ and diminishes over time. Figure 5(b) and (c) further display that the percentage of the quality improvement $(Q_{i+1} - Q_i/Q_i$ and investment increase $(I_{i+1} - I_i)/I_i$ are larger at the early stages of the investment and they diminish over time. In addition, Figure 5(d) displays seven investment elasticities with respect to seven starting results ranging from $S_1$ to $S_7$. It can be observed that starting from result $S_4$, the algorithm has the highest investment elasticity. More concretely, users

can obtain the highest percentage of quality improvement when they make the same percentage of cumulative investment increase if the algorithm starts from result $S_4$.

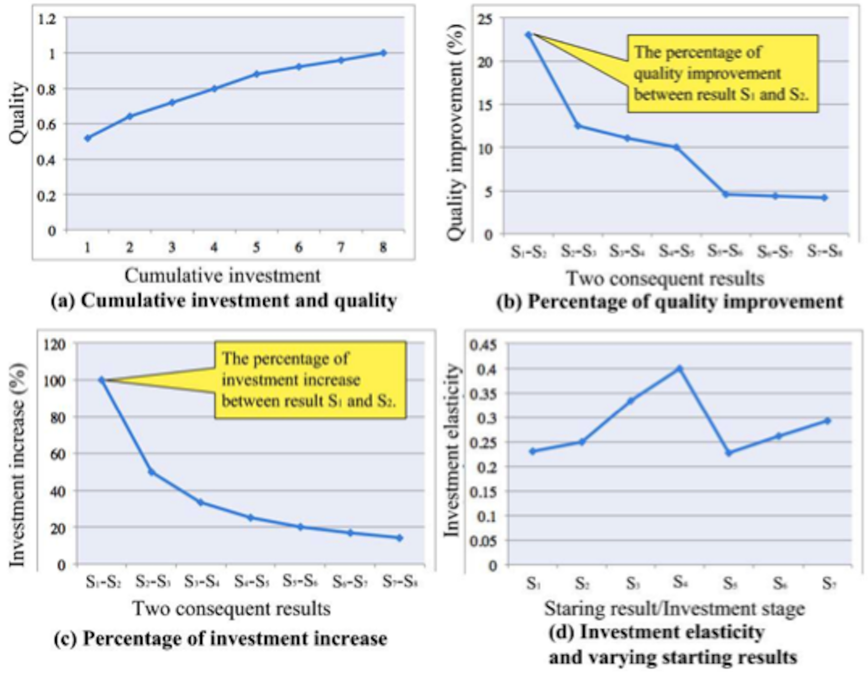



Figure. 5: Comparison of investment, quality and investment elasticity.

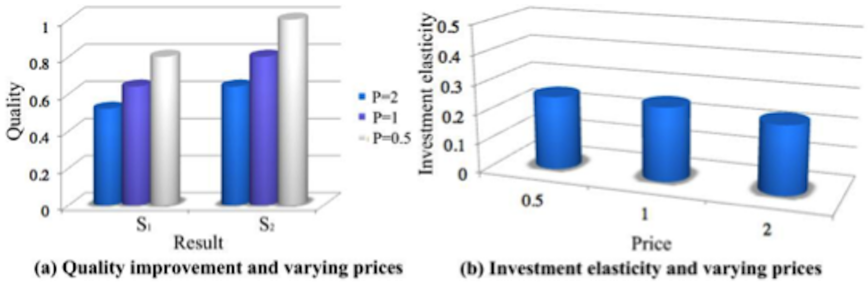



Figure. 6: An example investment elasticity under three prices.

Furthermore, suppose the investment function in a Cloud environment is $I = R \times P$. This means the investment is the product of the count of resource used, e.g. a virtual machine (VM) instance per hour and the unit price of the resource. According to the assumption that $\triangle I$ is fixed, we have the resource $\triangle R$ also fixed, which means the same amount of resource is needed to produce an improved result $S_{i+1}$ based on result $S_i$. Suppose that $\triangle R$ is running a standard small VM instance for an hour and the investment $\triangle I = 2$ dollars. Figure 6(a) shows that under three different prices $P = 0.5$, 1 and 2 (unit is dollar for running the standard small VM instance for an hour), the result has higher quality improvement at a lower price (more resources are consumed) and lower improvement at higher price. Furthermore, Figure 6(b) displays that given an EA, although the quality has different amount of prices with different prices, the algorithm has invariant investment elasticity with respect to varying prices. Thus, investment elasticity is a better criterion that characterizes the feature of the EA in the Cloud.

## 5. SUMMARY AND DISCUSSION

### 5.1 Summary

Cloud computing represents a paradigm shift in how we access and use computational resources. It is the realization that increasing or decreasing computing power can be delivered on-demand, that a pay-per-use model for using resources is now available and that the price of resources themselves may vary with time. The key challenge we face now is how to develop software programs that make use of such elasticity properties, i.e., how to make our algorithms themselves elastic.

In this paper, we proposed the concept of elastic algorithms for Cloud computing. We described a class of such algorithms that work by generating successive approximate results over large data sets and discussed their desirable properties. We also provided a formal definition of algorithmic elasticity and used it to investigate several forms of elasticity including investment elasticity, resource elasticity and price elasticity. We then described a generic approach for developing elastic algorithms over large data sets. The approach builds on using a hierarchical coding method to represent the available data at multiple levels of granularity that can be processed at different costs. We then provided an example of developing an elastic kNN classification algorithm using an R-tree data structure as the coding method. We investigated the quality monotonicity of the results using simple examples as well as investigated the other elasticity properties of the algorithm.

### 5.2 A Research Agenda for Elastic Algorithms

Our work in this paper focused on investigating the foundations and theoretical definitions of elastic algorithms and their properties. Clearly, our immediate future work is to investigate the practical implementation and evaluation of our kNN classification algorithm on real Cloud environments and under different pricing strategies. It also includes investigating applying the generic framework presented to other problems and investigating the use other data coding methods that can be applied for summarizing large data sets at multiple granularities, e.g., wavelet-based summarization. Making full use of the approach, however, requires addressing a number of key challenges that we summarize below.

*Challenge 1 Formal framework for reasoning about elasticity properties*: The major challenge when developing elastic algorithms is to ensure that the developed algorithm satisfies the desirable properties described in this paper. In particular, ensuring quality monotonicity as the computation proceeds is essential so that users do not waste their investment with no guarantees on return on their investment. This is in contrast to the anytime algorithm methodology, where users do not pay for resources and where once an acceptable result is produced users could keep the algorithm running until the deadline expires. Addressing this challenge requires the careful analysis of how quality varies with use of resources. For the kNN example presented in this paper we used the AUC metric as a quality measure and discussed, informally, how it is exhibits monotonic behaviour. Reasoning about monotonicity more rigorously requires the use of formal methods and proofs which are beyond the scope of this paper. Perhaps more challenging is not developing a proof for individual algorithms and quality functions, but rather developing a generic formal framework that can guide the development of quality-monotonic algorithms. This is an active area of our current research.

*Challenge 2 Scheduling elastic algorithms under budget and deadline constraints*: In this paper we did not investigate the problems associated with scheduling elastic algorithms to minimize their total execution costs while meeting user deadlines. We also did not investigate how to deal with resource price fluctuations used in dynamic pricing schemes. We note here that traditional resource scheduling algorithms are designed to produce the shortest execution schedule for a program on available resources, and assume that idle resources are free and the price of resources does not change with time and have no notion of result quality. Thus, in a *Pay-as-you-go* paradigm new scheduling algorithms need to be designed to take into consideration the properties

of both elastic algorithms and cloud environments. Our work provides a foundation for studying these problems and for investigating different strategies and algorithms. For example, one user may be more inclined to settle for an early result with adequate quality to save its available budget rather than waste it on diminishing returns of quality. In a spot price model another user may be willing to bid for resources at a higher price for earlier iterations to ensure that the computation produces an acceptable result before a deadline and then to bid for resources at a much lower price for extra improvements. Investigating such different scenarios and strategies is currently an active focus of our research, and we believe it could open a whole new area of research into scheduling computations on cloud environments.

*Challenge 3 Developing a reusable elastic algorithm development environment*: In this paper, we presented how to use hierarchical data structures (e.g. R-tree) with varying granularity to support the development of an elastic algorithm. In a practice, one would argue that developing and using such data structures from scratch may be beyond the capability of the average programmer. Going mainstream with the approach requires the development of a reusable algorithm development environment that provides programmers with a variety of data structures, programming libraries and associated tools that simplify the development of elastic algorithms. It would also require developing offline modeling tools and real-time performance monitoring that would support the users in profiling the behavior of their programs and in making investment decisions in real-time and also in evaluating the practicabilities of the proposed approach.

*Challenge 4 Applicability to wider problems*: The approach presented in this paper is, so far, a theoretical framework and is well suited for applications where users are willing to settle for approximate answers if the cost of generating the full results exceeds their available computation budget. As discussed in the paper, it naturally applies to a wide range of domains, including numerical, scientific and engineering computations, statistical estimation and prediction in data mining applications, heuristic search applications, database query processing applications and multimedia applications. However, many other applications may not lend them themselves easily to such a paradigm. For example, on the face of it, a traditional pay-roll application that calculates and transfers employees pay may not fit; paying half the employees only or approximating the salary of the employees could have detrimental consequences on the business itself. Even for such a pay roll application, one can see that the first result must perform a mandatory task that would be the initial result with minimum acceptable quality. One can also argue that further refinement iterations can produce associated management reports at different levels of granularity and quality. It would be interesting to investigate how the elastic algorithms approach can be applied effectively in such applications in practice.

## REFERENCES


Amazon Web Services (Amazon WS). *http://aws.amazon.com/ (14.06.13).*

DEAN, T., AND BODDY, M. 1988. An analysis of time-dependent planning. In *Proceedings of the seventh national conference on artificial intelligence AAAI, 49-54.*

DEAN, T.L. 1989. Intractability and time-dependent planning. In *Proceedings of the seventh national conference on artificial intelligence Los Altos, California (Morgan Kaufmann, 1987), 245-266.*

FAWCETT, T. 2006. An introduction to ROC analysis. In *Pattern recognition letters 27, 861-874.*

GABER, M.M., AND YU, P.S. 2006. A framework for resource-aware knowledge discovery in data streams: a holistic approach with its application to clustering. In *Proceedings of the 2006 ACM symposium on Applied computing (SAC '06) ACM, 649-656.*

GUO, YIKE, GHANEM, M. MOUSTAFA, AND HAN, RUI. 2012. Does the Cloud need new algorithms? An introduction to elastic algorithms. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on IEEE, 66-73.*

GUTTMAN, A. 1984. R-trees: a dynamic index structure for spatial searching. In *In Proceedings of ACM. 1984.*

RUI, HAN, GHANEM, M. MOUSTAFA, LI, GUO, YIKE, GUO, AND OSMOND, M. 2012. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. In *Future Generation Computer Systems. 2013.*

RUI, HAN, LI, GUO, GHANEM, M. MOUSTAFA, AND YIKE, GUO 2012. Lightweight Resource Scaling for Cloud Applications. In *Proceedings of the The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'12), Ottawa, Canada2012 IEEE, 644-651.*

Rui, Han, Li, Guo, Yike, Guo, and Sijin, He 2011. A Deployment Platform for Dynamically Scaling Applications in The Cloud. In *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom'11), Athens, Greece2011 IEEE, 506-510.*

HAND, D.J., and TILL, R.J. 2001. A simple generalisation of the area under the ROC curve for multiple class classification problems. In *Machine Learning 45, 171-186.*

HORVITZ, E. 1988. Reasoning about beliefs and actions under computational resource constraints. In *International Journal of Approx. Reasoning 2, 337-338.*

HORVITZ, E.J. 1990. Computation and Action under Bounded Resources. In *Department of EngineeringEconomic Systems stanford university, CA, 319.*

HWANG, K., DONGARRA, J., and FOX, G.C. 2012. Distributed and cloud computing. In *Elsevier/Morgan Kaufmann.*

LIU, J.W.S., LIN, K.J., SHIH, W.K., YU, A.C.S., CHUNG, J.Y., and ZHAO, W. 1991. Algorithms for scheduling imprecise computations. In *Computer 24, 58-68.*

MORENO-VOZMEDIANO., R., MONTERO., R.S, and LLORENTE, I.M. 2009. Elastic management of cluster-based services in the cloud. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds ACM, 19-24.*

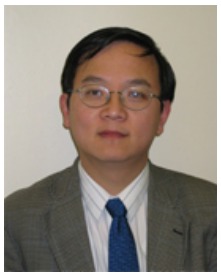
PHARR, M., and HUMPHREYS, G. 2004. Physically based rendering: From theory to implementation. In *Morgan Kaufmann.*

POLADIAN, V., SOUSA, J.P., GARLAN, D., and SHAW, M. 2004. Dynamic configuration of resource-aware services. In *Proceedings of 26th International Conference on Software Engineering (ICSE 2004) IEEE, 604-613.*

SAMUELSON., P.A., and NORDHAUS., W.A. 2011. Microeconomics. In *McGraw-Hill.*

SHARMA, U., SHENOY, P., SAHU, S., and SHAIKH, A. 2011. A cost-aware elasticity provisioning system for the cloud. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on IEEE, 559-570.*

WEBB, A.R., COPSEY, K.D., and CAWLEY, G. 2011. Statistical pattern recognition. In *Wiley. ZILBERSTEIN, S. 1996. Using anytime algorithms in intelligent systems. AI magazine 17, 73-83.*



**Yike Guo** is a professor in computing science at the Department of Computing, Imperial College London, UK. He graduated in computer science from Tsinghua University, PRC and received Ph.D. degree in computational logic and declarative programming at Imperial College London. He has been working in the area of data intensive analytical computing since 1995 when he was the technical director of Imperial College Parallel Computing Centre. During the last 10 years, he has been leading the data mining group of the department to carry out many research projects, including some major UK e-science projects such as discovery net on grid based data analysis for scientific discovery, MESSAGE on wireless mobile sensor network for environment monitoring, Biological Atlas of Insulin Resistance (BAIR) on system biology for diabetes study. He has been focusing on applying data mining technology to scientific data analysis in the fields of life science and healthcare, environment science and security. His research interests include large-scale scientific data analysis, data mining algorithms and applications, parallel algorithms, and cloud computing.

**Moustafa M. Ghanem** a research fellow at the Department of Computing, Imperial College London, UK. He holds a PhD and an MSc in high performance computing from Imperial College London. He is a Research Fellow in the Department of Computing, Imperial College London. His current research interests are in large-scale informatics applications, including large-scale data and text mining applications and infrastructures, Grid and Cloud computing and workflow systems for e-Science applications. He has published more than 80 papers in these areas. Moustafa Ghanem was previously Research Director at InforSense Ltd and VP for Research at Nile University, Egypt.

**Rui Han** is a researcher and PhD student at the Department of Computing, Imperial College London, UK. He received MSc from Tsinghua University, China. His research interests are cloud computing, cloud resource management and workflow technology. He has many practices at the design and development of cloud deployment platform and process-aware information system.