

# A Model of Computing as a Service for Cost Analysis

Kenneth Johnson<sup>1</sup> and John V. Tucker<sup>2</sup>

<sup>1</sup>Computer Science Research Group, Aston University,  
Birmingham, B4 7ET

`k.h.a.johnson@aston.ac.uk`

<sup>2</sup>Department of Computer Science, Swansea University,  
Swansea, Singleton Park, SA2 8PP

`j.v.tucker@swansea.ac.uk`

**Keywords:** cloud computing, abstract data types, abstract register machines, stream computation, cost analysis

**Abstract.** Cloud computing is a new technology paradigm in which computing power is delivered to the customer as a *service*. We develop a model of computing as a service to quantify monetary costs based on the time services are available and the volume of data they process.

## 1 Introduction

Cloud computing can be succinctly described as *computing as a service* [1,7,6], in which software, platforms and infrastructure are available from providers on a subscription basis such as pay-on-demand and pay-as-you-go. At the infrastructure level, *infrastructure services* are virtualised servers and storage devices purchased by the customer to host data and applications at the provider's data-centre. The most attractive feature of cloud computing is its elastic nature, which enables customers to adapt to suit variations in computing requirements over time, such as seasonal fluctuations.

Prominent among the attributes of computing as a service, at least to a customer of a service provider, is financial cost. This is determined by the infrastructure services selected by the customer, its deployment through service requests made via their account, and the use made of the services. More specifically, the cost of a computing service is determined by the

1. time duration the service is available (*service deployment*), and
2. volume of data handled by the service (*service usage*).

In this paper, we propose a model of computing as a service, focused on quantifying the monetary costs. The costs of computation are based on the two quantities, which are formulated as a function  $Cost(\pi, t) = Cost_D(\pi, t) + Cost_U(\pi, t)$  of the interaction  $\pi$  between the provider and the customer, where  $Cost_D(\pi, t)$  is the cost of service deployment at the customer's request and  $Cost_U(\pi, t)$  is the sum of costs of data handling across all services as scheduled by  $\pi$  over time  $t$ .

We proceed in two stages. First, we construct a model of the infrastructure available for rental. Each virtual infrastructure on offer can be approximated using the idea of an abstract register machine model with the following features:

- memory and storage for any form of data,
- programs with instructions based on general operations on the data, and
- non-terminating computation processing data in time.

Mathematically, the model is a form of register machine over a many-sorted algebra processing infinite streams of data from the algebra.

In the second stage we construct a model of the way a collection of computing services are chosen and made available over a time period. We then construct a model of the financial cost of these services. At the heart of our model is the formalisation of the dynamic nature of a customer’s account with a provider. Although deployment is not difficult to analyse, its integration with usage is subtle.

Throughout we use computing service offers from Amazon Web Services to guide and illustrate the construction of the cost model. However, the resulting model is a general algebraic model that is provider-independent.

A special technical feature of the discussion is the use of explicit clocks to model infrastructure offerings, index events and to measure time. The clocks are explicitly compared in different ways but the whole process of using a computing service is ultimately relative to physical *real-world time* as measured by a discrete clock  $T_W$ .

## 2 A Model of Infrastructure Services

**Time, Data and Streams** An abstract data type is modelled by a many-sorted  $\Sigma$ -algebra  $A$  comprising any non-empty sets of data, operations and tests [5,4]. To these operations, we add a norm  $\| - \| : A \rightarrow \mathbb{N}$  measuring the size of data. Time is represented as a discrete *clock* modelled by an algebra of the form  $T = (\{0, 1, \dots\}, 0, t + 1)$ . A stream is a sequence of data indexed by time. We represent a stream of data from the  $\Sigma$ -algebra  $A$  as a function of the form  $s : T \rightarrow A$ .

**Comparing Clocks** Clocks  $T$  and  $T'$  are related by means of various mappings. A surjective, monotonic function  $\lambda : T \rightarrow T'$ , is termed a *retiming*;  $T$  is said to be *faster* than  $T'$ . The *immersion* of  $\lambda$  is a function  $\bar{\lambda} : T' \rightarrow T$  defined by  $\bar{\lambda}(t') = \mu t[\lambda(t) = t']$  [2]. To relate events indexed by  $T$  with a real-world time clock  $T_W$ , we use a map  $\tau : T \rightarrow T_W$  called a *time schedule*;  $\tau$  induces a retiming.

**Abstract Machines** An infrastructure service is a stream transformer, processing an input stream timed by a clock  $T_{in}$  and generating an output stream timed by a clock  $T_{out}$ . We formalise infrastructure services as an *abstract machine* model  $\mathcal{M}$  of stream computation.

The *configuration*  $\gamma$  of  $\mathcal{M}$  comprises

1. a memory  $\rho : R \rightarrow A$  storing the data from  $A$  necessary to carry out computations in registers from the set  $R$ ,

2. input and output registers  $i$  and  $o$  modelling network ports containing the latest data value transmitted by the machine, and
3. a program  $pr$  and a pointer to the next instruction of  $pr$  to be executed.

Program instructions<sup>1</sup> are denoted by  $PI(\Sigma)$  and derived from the operations named by  $\Sigma$ . They include specialised instructions  $IN$  and  $OUT$  that perform the abstract machine's I/O processing.

Let  $AM$  denote the set of all abstract machine configurations. A single execution step of the program  $pr$  is performed by the function  $step : AM \rightarrow AM$  where  $step(\gamma)$  is the configuration after executing the next instruction indicated by  $\gamma$ .

**Input Data Stream** Over the course of a program's execution, data arrives on the input network port and is stored in the input register  $i$  for processing by the abstract machine. The arrival of a data value  $a \in A$  to an abstract machine with configuration  $\gamma$  is modelled by the function  $arr : AM \times A \rightarrow AM$  where  $arr(\gamma, a)$  is the updated configuration with the data value  $a$  loaded into  $i$ .

To model step-wise execution of an abstract machine program that accepts data values on an input networking port, we define the function  $next : AM \times A \rightarrow AM$  as the composition of the  $arr$  and  $step$  functions:

$$next(\gamma, a) = step(arr(\gamma, a)) \tag{1}$$

such that the data  $a$  is loaded into the input register and a single step of the program's execution is performed on the resulting machine configuration.

We suppose that data values supplied to the abstract machine are from the input stream  $u : T_{in} \rightarrow A$ , where new data values arrive on the machine's input network port at a rate determined by the clock  $T_{in}$ . Now, the *instruction clock*  $T_I$  counts the steps of the abstract machine's program execution. To model the arrival of data to the machine using Equation 1, we require a function  $\lambda : T_I \rightarrow T_{in}$  such that  $\lambda(n)$  is an index in  $T_{in}$  whereby the input register is loaded with the value  $u(\lambda(n))$  on the  $n^{th}$  step of the program's execution. We assume the function  $\lambda : T_I \rightarrow T_{in}$  is a retiming, i.e.  $T_I$  is faster than  $T_{in}$ . This essentially assumes a correctness condition stating that services are always capable of responding in a timely manner to data arriving on the input port<sup>2</sup>.

Computation over  $u$  with an initial machine configuration  $\gamma$  is defined by the function  $comp : AM \times [T_{in} \rightarrow A] \times T_I \rightarrow AM$ :

$$comp(\gamma, u, 0) = \gamma \tag{2}$$

$$comp(\gamma, u, n + 1) = next(comp(\gamma, u, n), u(\lambda(n + 1))). \tag{3}$$

In addition to the instruction clock  $T_I$  we require a system clock  $T_S$  that models the *speed* of the abstract machine. The machine cycles defined by  $T_S$  is given by a speed of (say)  $k$  Hertz. This determines a retiming  $\kappa : T_W \rightarrow T_S$  defined by  $\kappa(t) = \lfloor \frac{t}{k} \rfloor$ , where  $T_W$  is a real-time clock.

<sup>1</sup> For example, data transfer between registers,  $\Sigma$ -operation evaluation, and branching.

<sup>2</sup> Cloud computing customers try to ensure this condition is always satisfied using load balancers that distribute requests amongst their services. See concluding remarks.

We define the *instruction set retiming*  $\iota : AM \times [T_{in} \rightarrow A] \times T_S \rightarrow T_I$ . The relationship between the clocks is complex:  $T_S$  faster than  $T_I$  and instructions and data take different times. A concise formulation of  $\iota$  might arise from the introduction of execution-time constants for each instruction, or the size of the data arriving on the input stream, or from the way the machine instructions are implemented on  $A$ .

**Output Data Stream** Let  $u$  be an input stream and  $\gamma$  and configuration. The computations performed by the abstract machine generates an output stream, defined by the function  $G : AM \times [T_{in} \rightarrow A] \rightarrow [T_{out} \rightarrow A]$ , called an *output stream generator function*. The function  $G$  is defined by specifying the index of instructions of the program  $pr$  that transmit data on the output port of the machine. The times on  $T_{out}$  are formed by counting the occurrences of the *OUT* instruction over the course of a program’s execution. We define a search function  $\mu : AM \times [T_{in} \rightarrow A] \times T_{out} \rightarrow T_I$  in terms of Equations 2 and 3 where  $\mu(\gamma, u)(n)$  is the instruction number in  $T_I$  which transmits the  $n^{th}$  datum of the output stream. We have

$$\begin{aligned} \mu(\gamma, u)(0) &= \text{least } m[\text{instr}(\text{comp}(\gamma, u, m)) = \text{OUT}] \\ \mu(\gamma, u)(n + 1) &= \text{least } m > \mu(\gamma, u)(n)[\text{instr}(\text{comp}(\gamma, u, m)) = \text{OUT}], \end{aligned}$$

where  $\text{instr} : AM \rightarrow PI(\Sigma)$  returns the next instruction to execute. Now, the contents of the output register  $o$  at each clock cycle of  $T_{out}$  specifies the data value transmitted by the machine. Thus we define

$$G(\gamma, u)(n) = \text{output}(\text{comp}(\gamma, u, \mu(\gamma, u)(n) + 1)), \quad (4)$$

for all  $n \in \mathbb{N}$ , where  $\text{output}(\gamma)$  is the projection of output register  $o$  from  $\gamma$ .

A stream  $v : T_{out} \rightarrow A$  is the *generated output stream* of an abstract machine with initial configuration  $\gamma$  computing over input stream  $u : T_{in} \rightarrow A$  if, and only if,  $v(n) = G(\gamma, u)(n)$  for all  $n \in \mathbb{N}$ .

### Example: Modelling a Virtual Server as an Abstract Machine

The products that a service provider advertises describe infrastructure services in terms of the following attributes:

- hardware specifications, e.g., computation speed, memories and storage;
- software, e.g., data types and programs; and
- pricing details for usage of the service relative to a currency  $\mathcal{C}$ .

We consider an example based on a product description of a virtual server from the Amazon EC2 service provider<sup>3</sup> depicted in Figure 1. We show how the attributes contained in the description map to a concrete abstract machine model.

- An abstract machine modelling servers of this type requires two memories:
- $\rho_{ram} : \mathbb{Z}_{34.2\text{GB}} \rightarrow \{0, 1\}^{64}$ , models the random access memory component of the server, where the range of addresses and the length of the word stored at each address is determined by Figure 1(1,2) attributes respectively, and

<sup>3</sup> Complete product descriptions of Amazon EC2 virtual servers are found at: [aws.amazon.com/ec2/#instance](http://aws.amazon.com/ec2/#instance)

1. Memory: 34.2GB	3. Storage: 850GB	5. Compute Units: 13
2. Platform: 64-bit	4. Software: Windows	6. Price per Hour: 0.20¢

Fig. 1. Attributes of an Amazon EC2 virtual server.

- $\rho_{disk} : \mathbb{Z}_{850GB} \rightarrow A$ , models the hard disk of the server, where the storage capacity and its formatted capacity and the type of data  $A$  stored is determined by Figure 1(3,4) attributes respectively.

In Figure 1(4) the software attribute identifies the documented functionality of the product’s software, naming operations and data in an interface  $\Sigma$ . The software attribute specifies a program  $pr$  that comprises those instructions derived from  $\Sigma$ , and that are implemented by the  $\Sigma$ -algebra  $A$ .

In Figure 1(5) the attribute describes the hardware speed of the product in terms of Amazon’s *EC2 compute units*, defining the speed of the *system clock*  $T_S$  of the product. Typically, a provider supplies the customer with a mapping from proprietary units of CPU capacity to an equivalent value in terms of measurement in hertz. In our example, one EC2 compute unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. For  $T_W$  in seconds, the frequency of a one gigahertz system clock is  $10^9$  system cycles per second (say) on  $T_W$ , thus the linear retiming function  $\kappa : T_W \rightarrow T_S$  is  $\kappa(t) = \lfloor \frac{t}{13 \cdot 10^9} \rfloor$ .

### 3 The Computing Services Account

A computing services account is a record of all services that are currently deployed by the customer. We refer to these services as the customer’s *deployment*; an apparatus of infrastructure services that is managed by the customer who modifies services as necessary via their account.

Mathematically, let  $P$  be the non-empty finite set of products offered by a service provider. Suppose that each product can be modelled by an appropriate abstract machine  $\mathcal{M}_p$  as defined in §2 with machine configuration in  $AM_p$ . Let  $AM = \cup_{p \in P} AM_p \cup \{\perp\}$ , where  $\perp$  is a special element denoting the absence of a machine configuration. When a product  $p \in P$  is purchased, a new instance of a computing service is deployed and assigned a unique *identifier*  $k$  from the set  $ID$ . The deployed service  $p\text{-}k$  is the infrastructure service  $k$  of product type  $p$ , or simply *service*  $p\text{-}k$ , refers to an abstract machine with

- $\gamma_{p\text{-}k} \in AM_p$ , a configuration initialising service  $p\text{-}k$ ,
- $G$ , a stream generator function (4), and
- $T_I, T_S$  and  $\kappa, \iota$ , the clocks and clock relationships.

#### 3.1 Infrastructure Service Usage Specifications

By *service usage* we mean both the input data stream to an infrastructure service, and the generated output data stream from the service. Thus, mathematically, the usage of the service  $p\text{-}k$  is determined by the pair  $(u, v)$  of data streams where

$$v = G(\gamma_{p\text{-}k}, u). \tag{5}$$

Now, to reason about service usage in real-time, we associate with  $u$  a time schedule function  $\alpha : T_{in} \rightarrow T_W$ . Shortly, we deduce a time schedule function  $\beta : T_{out} \rightarrow T_W$  for the output stream  $v$  as follows. Let  $\theta : T_I \rightarrow T_W$  be a function where  $\theta(n)$  is the time in  $T_W$  such that the machine  $\mathcal{M}$  modelling the service has completed the  $n^{th}$  instruction of the program contained in its configuration. We set  $\theta(n) = \bar{\kappa}(\bar{\iota}(\gamma_{p.k}, u)(n + 1))$  where we use  $\bar{\iota}(\gamma_{p.k}, u) : T_I \rightarrow T_S$  to return first system cycle  $s \in T_S$  of the *next* instruction  $n + 1 \in T_I$ , and the immersion  $\bar{\kappa} : T_S \rightarrow T_W$  to map  $s$  to the real-world time when it occurred. Thus, we set

$$\beta(n) = \theta(\mu(\gamma_{p.k}, u)(n)). \quad (6)$$

### 3.2 Starting and Stopping Services

We model the *state* of a customer's account as a function of the form  $\sigma : P \times ID \rightarrow AM$  where each product type  $p \in P$  and identifier  $k \in ID$  is associated with a machine configuration in  $AM$  and  $S = [P \times ID \rightarrow AM]$  denotes the set of all customer account states.

The starting of a service  $p.k$  is modelled by the function  $start_p : ID \times S \rightarrow S$  defined as

$$start_p(k, \sigma)(p.k') = \begin{cases} \gamma_{p.k} & \text{if } p.k = p.k' \\ \sigma(p.k') & \text{otherwise} \end{cases}$$

starting the abstract machine's computation immediately.

The termination of service  $p.k$ , is modelled by the function  $stop_p : ID \times S \rightarrow S$  defined as

$$stop_p(k, \sigma)(p.k') = \begin{cases} \perp & \text{if } p.k = p.k' \\ \sigma(p.k') & \text{otherwise,} \end{cases}$$

resulting in  $p.k$  ceasing to exist, its computations and data destroyed. References to service  $p.k$  result in the absent machine configuration  $\perp$ .

### 3.3 Service Deployment Specification

The provider supplies a set of operation names  $OP = \{start_p(k), stop_p(k) \mid p \in P, k \in ID\}$  which enable the customer to modify their service deployment. Operations are applied to an account state  $\sigma$  by the function  $apply : S \times OP \rightarrow S$  defined as

$$apply(\sigma, op) = \begin{cases} start_p(k, \sigma) & \text{if } op = start_p(k) \\ stop_p(k, \sigma) & \text{if } op = stop_p(k) \end{cases}$$

for a state  $\sigma \in S$  and operation  $op \in OP$ .

**Dynamic Behaviour of Customer Accounts** A customer's usage of their account is modelled by an *operation stream*  $\delta : T_E \rightarrow OP$  indexed by the event clock  $T_E$ .

Let  $\sigma_0$  denote the initial state such that  $\sigma_0(p.k) = \perp$ , for all  $p.k \in P \times ID$ . To determine the dynamic behaviour of the customer's account as operations from  $\delta$  are performed, we define  $Apply : S \times [T_E \rightarrow OP] \times T_E \rightarrow S$  where

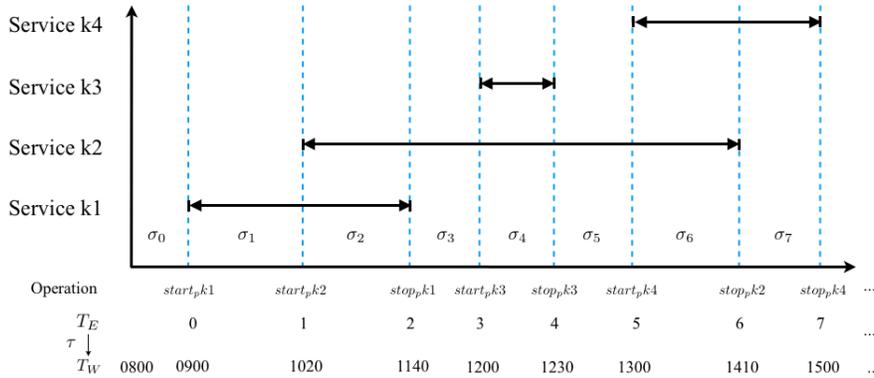
$$Apply(\sigma_0, \delta, 0) = \sigma_0$$

$$Apply(\sigma_0, \delta, n + 1) = apply(Apply(\sigma_0, \delta, n), \delta(n)).$$

An operation stream  $\delta$  induces a stream  $(\sigma_n)$  of account states indexed by the clock  $T_E$ . Denoting  $\sigma_{n+1} = Apply(\sigma_0, \delta, n)$  we observe that account state  $\sigma_{n+1}$  is created by the application of operation  $\delta(n)$ . We call  $(\sigma_n)$  the  $\delta$ -induced state stream.

### 3.4 Real-Time Deployment Specifications

Figure 2 presents a Gantt chart representation of a typical deployment scenario beginning at 0800h (when the account was created) until 1500h. The customer performs operations indexed by  $T_E$  that start and stop four services of the same product type  $p$  and identified as  $k_1, k_2, k_3$ , and  $k_4$ . The sequence of states induced as a result of the operations are demarcated by dashed vertical lines and are labelled, starting with the initial state  $\sigma_0$ . The service deployment of each  $k_i$ , for  $1 \leq i \leq 4$  is depicted by horizontal arrowed lines. Each operation has been associated with a time stamp recording the time it was performed.



**Fig. 2.** Gantt chart depicting account activity between 0800 and 1500

Mathematically, we define a time schedule  $\tau : T_E \rightarrow T_W$  such that  $\tau(n)$  is the time in  $T_W$  that operation  $\delta(n)$  is performed. For example, in Figure 2,  $\tau(2) = 1140h$  and  $\tau(6) = 1410h$ .

We call the pair  $\pi = (\delta, \tau)$  a *deployment plan*. Let  $Plan$  denote the set of all deployment plans. We define two important functions:

1.  $d : Plan \times T_E \rightarrow T_W$  calculates the duration of the  $n^{th}$  state of the  $\delta$ -induced state stream of the deployment plan  $\pi$ :

$$d(\tau, n) = \begin{cases} \tau(n) & \text{if } n = 0 \\ \tau(n) - \tau(n - 1) & \text{otherwise.} \end{cases}$$

2.  $duration : P \times ID \times Plan \times T_E \rightarrow T_W$  calculates duration of a service  $p.k$  deployment over all states of the  $\delta$ -induced state sequence of  $\pi$ :

$$duration_{p.k}(\pi, 0) = 0$$

$$duration_{p.k}(\pi, n + 1) = \begin{cases} duration_{p.k}(\pi, n) + d(\tau, n + 1) & \text{if } on_{p.k}(\sigma_{n+1}), \\ duration_{p.k}(\pi, n) & \text{if } \neg on_{p.k}(\sigma_{n+1}) \end{cases}$$

where the *deployment predicate*  $on : P \times ID \times S \rightarrow \mathbb{B}$  determines the deployment status of a service  $p.k$  and is defined as  $on_{p.k}(\sigma) = (\sigma(p.k) \neq \perp)$ .

## 4 Cost Modelling and Analysis

### 4.1 Service Deployment Costing

For a provider offering products from  $P$ , we formalise the billing model as a  $P$ -indexed family of functions  $Bill_P = \langle bill_p : T_W^n \rightarrow \mathcal{C} \mid p \in P \rangle$ . Each function  $bill_p$  works out the *deployment cost* of services of product  $p$  in terms of a currency  $\mathcal{C}$  based on real-time information such as the time the service was started and how long it has been deployed.

**Example: Amazon EC2 Billing Model** For  $T_W$  measured in minutes and a deployment duration  $D$ , the Amazon EC2 billing model for a product type  $p$  is a function  $bill_p : T_W \rightarrow \mathcal{C}$  defined as  $bill_p(D) = \lceil \frac{D}{60} \rceil \cdot price$ , the number of hours the service is deployed rounded up to the nearest hour and scaled according to the unit price of  $p$  in currency  $\mathcal{C}$ .

To enable us to apply billing models to services over physical time intervals measured in terms of  $T_W$ , we define a function  $\epsilon : [T \rightarrow T_W] \rightarrow [T_W \rightarrow T]$  such that given a time schedule  $\tau : T \rightarrow T_W$  and time  $t$  in  $T_W$ ,  $\epsilon_\tau(t)$  is the index in  $T$  of the next event occurring after  $t$ . We define  $\epsilon_\tau(t) = \min\{n \in T \mid t < \tau(n)\}$  and call  $\epsilon_\tau$  the *event function* of  $\tau$ .

When  $T = T_E$ ,  $\epsilon_\tau(t)$  is the index of the account state in the  $\delta$ -induced state sequence  $(\sigma_n)$  of  $\pi$  at time  $t^4$ .

We define the function  $cost_p : ID \times Plan \times T_W \rightarrow \mathbb{N}$  which calculates the deployment cost of service  $p.k$  on  $\pi$  as

$$cost_p(k, \pi, t) = bill_p(duration_{p.k}(\pi, \epsilon_\tau(t)) - [\tau(\epsilon_\tau(t)) - t]).$$

In order to calculate the combined costs of all services deployed by the deployment plan  $\pi$ , we define the subset  $V_n \subseteq P \times ID$  as  $p.k \in V_n \iff on_{p.k}(\sigma_n)$ , containing the set of all references of deployed services in state  $\sigma_n$ . The total cost of deploying services over  $\pi$  is calculated by the function  $Cost_D : Plan \times T_W \rightarrow \mathbb{N}$  such that

$$Cost_D(\pi, t) = \sum_{p.k \in V_{\epsilon_\tau(t)}} cost_p(k, \pi, t).$$

<sup>4</sup> For example,  $\epsilon_\tau(1100h) = 2$  and  $\epsilon_\tau(1410h) = 7$  for  $\tau$  in Figure 2.

## 4.2 Service Usage Costing

Table 1 presents a five-tiered pricing scheme based on prices for transferring data out of Amazon’s EC2 Cloud. Each numbered price tier comprises values specifying a data quota, e.g. the maximum amount of data to be costed at that tier, and the unit price per gigabyte.

**Table 1.** A Five-Tiered Pricing Scheme for Data Transmission.

Tier	1	2	3	4	5
Quota	[0, 10]	(10, 40]	(40, 100]	(100, 350]	> 350
Price(\$)	0.120	0.090	0.070	0.050	0.040

Mathematically, we model an  $N$ -tiered pricing scheme as a finite set  $\Phi = \{(pr_i, qo_i, t_i) \mid 1 \leq i \leq N\}$ . For each  $i \in \mathbb{N}$  the values  $pr_i \in \mathcal{C}$  and  $qo_i \in \mathbb{N}^\top$  model the unit price in the currency  $\mathcal{C}$  and the data quota of the  $i^{\text{th}}$  tier respectively, and  $t_i : \mathbb{N} \rightarrow \mathbb{N}$  is a function defined by

$$t_i(z) = \begin{cases} z \cdot pr_i & \text{if } z < qo_i, \\ qo_i \cdot pr_i & \text{otherwise,} \end{cases}$$

governs the application of the  $i^{\text{th}}$  tier price to a data amount  $z$  in  $\mathbb{N}$ . For the  $N^{\text{th}}$  tier of  $\Phi$ , the quota  $q_N$  is set to a special element  $\top$  with the properties  $z < \top$  and  $z \boxminus \top = 0$  for all  $z \in \mathbb{N}$ , where  $\boxminus$  is the cut-off subtraction operator. Let *Scheme* denote the set of all finite-tiered pricing schemes.

The cost of transferring data of size  $z$  relative to an  $N$ -tiered pricing scheme  $\Phi$  is calculated by the function  $Tier : \text{Scheme} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{C}$  defined as

$$Tier(\Phi, z, 0) = 0$$

$$Tier(\Phi, z, n + 1) = \begin{cases} Tier(\Phi, z, n) & \text{if } n + 1 > N, \\ t_{N-n}(z) + Tier(\Phi, z \boxminus qo_{N-n}, n) & \text{otherwise.} \end{cases}$$

Tiered pricing schemes can be applied to perform cost analysis on the data carried by a stream<sup>5</sup>. We define the function  $usage : [T \rightarrow A] \times \text{Scheme} \times \mathbb{N} \times T \rightarrow \mathbb{N}$  as  $usage(s, \Phi, N, n) = Tier(\Phi, Size(s, n), N)$  which calculates the usage cost up to and including the  $n^{\text{th}}$  data value carried by the stream  $s : T \rightarrow A$  and applies the  $N$ -tiered pricing scheme  $\Phi$ .

In order to work out the cost of service usage accord all deployed services, we suppose that a service  $p.k$  is associated with an input data stream  $u_{p.k} : T_{in}^{p.k} \rightarrow A$  carrying the data arriving as input to the service  $p.k$  according to a time schedule  $\alpha_{p.k} : T_W \rightarrow T_{in}^{p.k}$ . By Equation (5) we generate the output data stream  $v_{p.k} : T_{out}^{p.k} \rightarrow A$  of service  $p.k$  and use Equation (6) to deduce its time schedule  $\beta : T_W \rightarrow T_{out}^{p.k}$ .

We are now in a position to work out the cost of the usage for services deployed over the course of the deployment plan  $\pi$  each supplied with a data

<sup>5</sup> To encourage migration to the cloud, some providers do not charge for data input.

input stream. We suppose that the cost of data input is determined by an  $N$ -tiered pricing scheme  $\Phi_I$ , and the cost of the generated output data is determined by an  $M$ -tiered pricing scheme  $\Phi_O$ . The function  $Cost_U : Plan \times T_W \rightarrow \mathcal{C}$  is defined as

$$Cost_U(\pi, t) = \sum_{p.k \in V_{\epsilon_\tau(t)}} usage_p(u_{p.k}, \Phi_I, \epsilon_\alpha^{in}(t), N) + usage_p(v_{p.k}, \Phi_O, \epsilon_\beta^{out}(t), M)$$

where  $\epsilon_\tau$ ,  $\epsilon_\alpha^{in}$  and  $\epsilon_\beta^{out}$  are event functions from time schedules  $\tau$ ,  $\alpha$  and  $\beta$ .

Thus, the total cost is calculated by  $Cost : Plan \times T_W \rightarrow \mathcal{C}$  and is defined as

$$Cost(\pi, t) = Cost_D(\pi, t) + Cost_U(\pi, t).$$

## 5 Concluding Remarks

Our model addresses the need for a precise cost analysis of computing as a service. A key feature is the costing of requirements based on customer interaction with the provider. In [3] this was tackled using model checking techniques for service usage expressed as probabilistic patterns. Our model of the services themselves is rudimentary, but has served to identify an important *performance requirement* condition (§2). Future work will address performance issues which are central to both customers and providers. It will involve modelling more complex services such as load balancers and auto-scalers that work together to generate deployment plans based on specified performance conditions.

## References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. *Commun. ACM* 53, 50–58 (2010)
2. Harman, N., Tucker, J.V.: Algebraic models of microprocessors architecture and organisation. *Acta Informatica* 33, 421–456 (1996)
3. Johnson, K., Reed, S., Calinescu, R.: Specification and quantitative analysis of probabilistic cloud deployment patterns. In: *Haifa Verification Conference 2011*. LNCS, Springer (to appear)
4. Meinke, K., Tucker, J.V.: Universal Algebra, pp. 189–411. *Handbook of Logic for Computer Science*, Oxford University Press (1992)
5. Tucker, J.V., Zucker, J.I.: Computable functions and semicomputable sets on many sorted algebras, pp. 317–523. *Handbook of Logic for Computer Science. Volume V Logic and Algebraic Methods*, Oxford University Press (2000)
6. Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.* 39, 50–55 (2008)
7. Youseff, L., Butrico, M., Da Silva, D.: Toward a unified ontology of cloud computing. In: *Grid Computing Environments Workshop (GCE '08)*. pp. 1–10 (2008)

---

**Acknowledgements:** This work was partly supported by the UK Engineering and Physical Sciences Research Council grant EP/H042644/1.

We thank Radu Calinescu for helpful comments on an earlier draft of this paper.