Specification and Quantitative Analysis of Probabilistic Cloud Deployment Patterns

Kenneth Johnson, Simon Reed and Radu Calinescu

Computer Science Research Group, Aston University, Birmingham B4 7ET UK
{k.h.a.johnson, reeds, r.c.calinescu}@aston.ac.uk

The probable is what usually happens.

Aristotle

Abstract. Cloud computing is a new technological paradigm offering computing infrastructure, software and platforms as a pay-as-you-go subscription based service. Many potential customer's of cloud services require essential cost assessments to be undertaken before transitioning to the cloud. Current assessment techniques are imprecise and rely on simplified resource requirements that fail to account for probabilistic variations in usage. In this paper, we address these problems and propose a new *probabilistic pattern modelling* (PPM) approach to cloud costing and resource usage assessment. Our approach is based on a concise expression of probabilistic resource usage patterns translated to *Markov decision processes* (MDPs). Key costing and usage queries are identified and expressed in a probabilistic variation of temporal logic and calculated to a high degree of precision using quantitative verification techniques. The PPM cost assessment approach has been implemented as a Java library and validated with a case study and scalability experiments.

1 Introduction

Cloud computing can be succinctly described as *computing as a service* [1,23,20] where software, platforms and virtualised hardware are available on-demand on a pay-as-you-go basis from a cloud provider. The elastic nature of the cloud enables customers to adapt service usage to meet fine-grained variations of their resource requirements by dynamically scaling their computing services up or down. This situation seems economically favourable in comparison to making large initial investments on infrastructure based on requirements for peak demand. Despite the envisioned benefits of cloud computing, there are still barriers to quick adoption of cloud services. Many potential customers are still reticent of cloud computing due to an inability to accurately express and analyse their resource requirements, alongside other concerns such as cloud security[10], reliability and compliance with data protection laws [11].

A precise expression of resource requirements is an important step in deciding beforehand the cost-savings of transitioning to the cloud. With concise

requirements specifications, customers could readily make accurate comparisons between several potential cloud deployments in terms of available capital. This would enable cloud providers offering identical services (for example, virtual machines with the same computing capabilities) to be easily evaluated on the basis of the cost of deployment.

The most attractive feature of cloud computing services is the ability to dynamically scale resources up or down over fine-grained time intervals. As a result, cloud requirements are often thought about in terms of *patterns of usage*, where resource requirements vary over time. Resource usage can change due to small variances in workload or changing economic situations such as fluctuations in a cloud provider's prices and the customer's capital income.

These types of resource usage patterns are inherently *probabilistic* in nature and involve potentially unknown or *non-deterministic* factors, making requirements specification difficult and applying existing cost assessment methods less accurate. Current cost modelling tools such as [13] and [3] model requirements with usage patterns that do not take into account the probabilistic nature of resource usages resulting from the cloud's dynamic scalability. This leads to a naive cost assessment, where probabilistic behaviours do not play a role in determining cost. Instead, resource usage is simplified to follow either constant rates of change or variations over coarser time intervals. To overcome these limitations we propose a new approach for the expression of resource requirements as *probabilistic patterns*, and the application of *quantitative verification* techniques to analyse a wide range of cost-related characteristics of potential cloud deployments.

Probability has been used to model unreliable or unknown behaviour in both hardware and software systems, and thanks to the development of powerful software tools PRISM[16], MRMC[12] and RAPTURE[9], quantitative verification has found applicability in a wide range of application domains: verification of QoS properties in service-based systems[5], run-time model checking to guide self-optimisation strategies in software systems [6,4] and most recently in [15] where probabilistic modelling was used for performance analysis of live migration of virtual machines between physical servers in a cloud data center.

Using the mathematical techniques of quantitative verification [19,18], our approach enables potential customers of cloud services to check quantitative properties of a cloud deployment that are broadly categorised into

- **costs:** to determine a deployment's variation of costs over time, and to calculate the maximum accumulated costs owed to a cloud provider at the end of a billing period, and
- **resource usage:** to determine the maximum and minimum probabilities that a deployment's resource usage exceeds a certain threshold,

specified in rewards-augmented probabilistic computational tree logic.

The main contributions of our work are:

1. A high-level language for the specification of probabilistic and non-deterministic patterns of cloud resource usage,

2. Techniques to synthesise Markov decision process (MDP) models from resource usage patterns and to formalise resource usage and cost properties as reward-augmented PCTL formulae [8].
3. An implementation of our approach as an open-source Java package, and
4. A case study and scalability experiments to validate the approach.

The paper is organised as follows: Section 2 provides background information on Markovian models, property specification and probabilistic model checking. Section 3 presents the steps of the technique in detail. A grammar for probabilistic patterns is given and we describe an algorithm for MDP synthesis. Properties for cost and resource usage analysis are formalised as probabilistic temporal logic. Section 4 introduces the prototype tool `PPM` (Probabilistic Pattern Modeller) implementing the approach and presents a realistic case study and scalability experiments. Section 5 discusses related results and Section 6 summarises our results and suggests directions for future work.

2 Background

A *Markov decision process* (MDP) is a tuple

$$M = (S, s_0, Act, Dist, step, L)$$

where $S = \{s_0, \ldots, s_n\}$ is a finite set of states, s_0 the initial state, Act a set of actions, $Dist$ a set of probabilistic distributions over the states in S, $step : S \to 2^{(Act \times Dist)}$ is a probabilistic transition function that maps the states in S to finite sets of actions-distributions pairs, and $L : S \to 2^{AP}$ is a map labelling states with finite sets of atomic properties in AP.

Probabilistic model checking [18] is a technique for building specifications of systems exhibiting probabilistic behaviour and determining the satisfiability of quantitative properties. Properties are specified in a probabilistic temporal logic PCTL extending CTL with a probabilistic operator $P_{\bowtie p}$, for $p \in [0, 1]$ and $\bowtie \in \{<, \leq, \geq, >\}$. A wide-range of quantitative properties for Markovian models can be specified in this logic. For example "the probability of a cloud deployment eventually requiring x or more resources is less than p" can be specified by the PCTL formula $P_{<p}[F\ res \geq x]$. For model checking MDPs, the operators $Pmin_{\bowtie p}$ and $Pmax_{\bowtie p}$ determine the minimum and maximum probabilities over all adversaries (e.g. all resolutions of non-determinism), respectively.

A *reward structure* for an MDP assigns data to states and transitions that are interpreted, for example, as resource usage. Formally, a rewards structure is a pair of functions (r_s, r_a) such that $r_s : S \to \mathbb{R}_{\geq 0}$ is a state-reward function, and $r_a : S \times S \times Act \to \mathbb{R}_{\geq 0}$ is a transition-reward function. PCTL is extended to include rewards-augmented operators: instantaneous rewards $R_{\bowtie r}[I^{\leq t}]$ and cumulative rewards $R_{\bowtie r}[C^{\leq t}]$.

Our work uses the probabilistic model checker PRISM [16] to automatically verify quantitative properties of Markovian models expressed in PRISMs high

level state-based language. Our choice of PRISM was motivated by recent improvements in the software's MDP verification[17] algorithms and expressive property specification language, supporting (probabilistic) temporal logics and rewards-augmented extensions.

3 Approach

Our approach to assessing the cost and resource usage characteristics of cloud deployments (Figure 1) comprises the following steps:

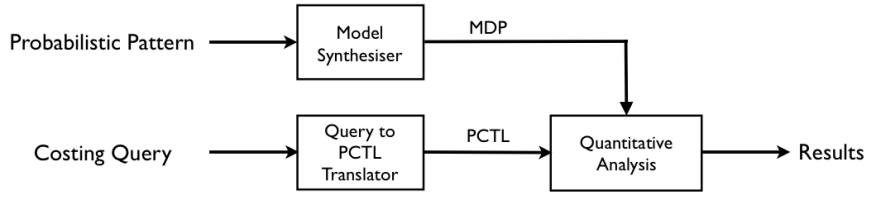

Fig. 1. Quantitative analysis of cloud deployments

1. Specification of the resource requirements for the cloud deployment as a high-level probabilistic pattern.
2. Generation of MDP: patterns from Step 1 are accepted as input to a *model synthesiser* that outputs an MDP model formalising probabilistic resource variations over a time interval associated, for example, with a particular day, week, or billing period.
3. Queries relating to costs or other quantitative properties of the cloud deployment act as input to the *query to PCTL translator* that outputs the query formalised as a PCTL formula.
4. The MDP model generated in Step 2 and the PCTL formula from Step 3 act as input to a probabilistic model checker which performs quantitative analysis. The numerical results of the analysis are output to the customer for further analysis or data visualisation.

3.1 A Language for Probabilistic Patterns

Resource requirements of a cloud deployment are formalised by probabilistic patterns, expressed in terms of a simple language[1] based on the declaration of rules to specify elastic variations of cloud resource usage over time.

In this section we develop a language for expressing probabilistic and non-deterministic behaviours of cloud requirements.

Formally, a *probabilistic pattern*

$$P = BR^* \tag{1}$$

[1] We do not expect probabilistic patterns to be composed by hand but rather generated automatically from log analysis of application resource usage and requests.

is a high-level syntactic representation of a customer's hourly usage of cloud computing resources beginning with a baseline declaration B, and optionally followed by a finite list of r rule declarations $R_1, \ldots, R_r$. We list the syntactic form of each declaration in the language with terminals displayed in `fixed-width font`.

A baseline declaration B has the syntactic form

$$\texttt{Baseline } b \tag{2}$$

specifying that the user requires a constant amount b of cloud computing resources (e.g. virtual machines).

A rule declaration R has the syntactic form

$$\texttt{Rule Start } S \texttt{ Vary } V \tag{3}$$

which enables the specification of a variation V from the constant baseline amount of resource usage at a certain point in time S. For example a rule might specify an increase in resources to meet computing requirements at known peak times when more resources are needed. Such variations are probabilistic in nature to reflect the increase and decrease of resources depending on factors such as workload, capital, economies, and even the fluctuations in the pricing of cloud services themselves.

A variation V is a set of m discrete probability distributions

$$\{D_1, \ldots, D_m\} \tag{4}$$

each of which has the syntactic form

$$\texttt{[}p_1\texttt{:}Op_1\texttt{(}z_1\texttt{) + ... + }p_q\texttt{:}Op_q\texttt{(}z_q\texttt{)]} \tag{5}$$

to express the fact that change in resources involves

- a non-deterministic choice of a probability distribution $D_i \in V$, and
- and a selection of a resource usage *variant* $Op(z)$ according to the probability distribution D, with probabilities $p_1, p_2, \ldots, p_q$, $\sum_{j=1}^{q} p_j = 1$.

A variant $Op(z)$ comprises a name Op of an operation and a numerical operand z. Variants perform arithmetical operations on the resource amount res at time S to yield a new resource usage value res' for time $S+1$. The types of variants that can be used in patterns are given in Table 1, noting the variant `bl` does not require an operand.

Example 1. Suppose that a cloud deployment associated with a set of applications whose resource requirements follow a weekly (probabilistic) pattern. Assume for instance, that less resources are required at the weekend than during the week. We formulate these requirements by the probabilistic pattern

```
Baseline 10
Rule1 Start Jun 4th 0h  Vary {[0.6:sub(2) + 0.4:sub(5)],
                              [0.1:add(1) + 0.9:sub(3)]}
Rule2 Start Jun 5th 23h Vary {[1:bl]}
```

Table 1. Pattern Variants

Variant $Op(z)$	**Description**	**Resources** res' **at time** $S+1$
`add(`z`)`	Apply arithmetical operation to current resource usage amount	$res + z$
`sub(`z`)`		$res - z$
`mult(`z`)`		$res \times z$
`div(`z`)`		$res \div z$
`bl`	Set resource usage to baseline value	b
`bl-add(`z`)`	Set resource usage to baseline value and apply arithmetical operation.	$b + z$
`bl-sub(`z`)`		$b - z$
`bl-mult(`z`)`		$b \times z$
`bl-div(`z`)`		$b \div z$

that declares a baseline of 10 resource units followed by two rules. The first rule starts at the beginning of Saturday, June 4^{th} (2011), varying resource usage according to the two stated probabilistic distributions. Each of the variants in the distributions subtract resources from the baseline amount using the variant `sub`, suggesting that this rule generally decreases resource usage, except for a small probability of selecting the variant `add(1)`. The second rule begins at the end of Sunday June 5^{th}, and consists of a single distribution setting the resource usage back to the baseline amount using the variant `baseline`.

Of course, the user might want to specify that less resources are required for *every* weekend during certain months in the year. To support this scenario, our language allows the specification of an optional repeat declaration for each rule declaration 3:

$$\texttt{Rule Start } S \texttt{ Vary } V \texttt{ Repeat F Until U} \tag{6}$$

where F is a keyword from the set $\{\texttt{Day}, \texttt{WeekDay}, \texttt{Week}, \texttt{WeekEnd}, \texttt{Month}\}$ setting the frequency of the rule's application, and U specifies the last time the rule is to be applied.

Example 2. Using the repeat declaration in Line 6 to extend Example 1 gives the probabilistic pattern

```
Baseline 10
Rule1 Start Jun 4th 0h  Vary {[0.6:sub(2) + 0.4:sub(5)],
                              [0.1:add(1) + 0.9:sub(3)]}
                        Repeat WeekEnds Until Aug 31
Rule2 Start Jun 5th 23h Vary {[1:bl]}
                        Repeat Weekends Until Aug 31
```

which specifies that both rules apply every weekend during the summer months of June, July and August.

3.2 Markov Decision Process Synthesis

The model synthesiser takes as input a probabilistic pattern $P = BR^*$ in the form of the concrete syntax described in Section 3.1, and a time interval

$$T = [0, n] \tag{7}$$

and outputs an MDP model that allows the formal analysis of cost- and resource-related characteristics of the pattern.

We say an MDP M *models* P over T if it satisfies the following properties:

1. The state space is $S = \{(res, t) \mid res \in [0, Max], t \in T\}$, for $Max > 0$ associating each time index t a value res representing a resource amount.
2. The initial state is $(b, 0)$ according to the value b specified in the baseline declaration B.
3. Each state transition is of the form $(res, i) \xrightarrow{p} (res', i + 1)$ where p is the probability of requiring the resource amount res' at time $i+1$, for $0 \leq i < n$.

To determine resource usage change according to the variants, we define the function $next : S \times Variant \to S$, such that given a state $(res, t) \in S$ and a variant $Op(z)$, $next(s, Op, z)$ is the next state after applying the variant:

$$next((res, t), Op, z) = tick((res', t)),$$

where res' is the new resource amount from application of the variant (c.f. Table 1) and the function $tick : S \to S$ updates time t by one hour, defined by

$$tick(res, t) = (res, t + 1).$$

We denote the subset of all states in S associated with time t as S^t and note that this forms a partition of the state space S, where $S^t \cap S^{t'} = \emptyset$, for $t \neq t'$ and $S = S^0 \cup S^1 \cup \cdots$. The set S^{t+1} is defined inductively by the equation

$$S^{t+1} = \begin{cases} \bigcup_{D \in V} \bigcup_{p:Op(z) \in D} \\ \{next(s, Op, z) \mid s \in S^t\} & \text{if rule(s) in } P \text{ start at time } t \\ \\ \{tick(s) \mid s \in S^t\} & \text{otherwise} \end{cases} \tag{8}$$

where $S^0 = \{s_0\}$ contains the initial state. The states in S^{t+1} are calculated by applying variants $Op(z)$ (over all probability distributions in V) to the states in S^{t+1} whenever a rule is applied at time t. If no rule is applied, only the time is updated by the *tick* function.

Similarly, the transition function *step* is defined inductively by the equation

$$step(s) = \begin{cases} \bigcup_{d \in D} \bigcup_{p:Op(z) \in d} \\ \{s \xrightarrow{p} next(s, Op, z) \mid s \in S^t\} & \text{if rule(s) in } P \text{ starts at time } t \\ \\ \{s \xrightarrow{1} tick(s) \mid s \in S^t\} & \text{otherwise.} \end{cases} \tag{9}$$

where $s \xrightarrow{p} next(s, Op, z)$ are transitions from states in S^t to the states in S^t, associated with the probability p of making the transition. If no rule is applied then states in S^t transition to states in S^{t+1} with a probability of 1.

3.3 Quantitative Analysis of Cloud Deployment Queries

Using the query to PCTL translator, customers can input high-level queries relating to cost and other quantitative properties to obtain the query formalised as a rewards-augmented PCTL formula.

In this section, we focuses on two categories of properties deemed most relevant to a customer analysing a prospective cloud deployment: resource usage and cost. We present a list of queries, their specification in PCTL and outline the verification of each property on an MDP pattern model M over time interval (7). The high-level resource usage query

1. *What is the maximum probability of the cloud deployment's resource requirements equalling or exceeding the amount x?*

is specified by the PCTL formula $Pmax_{=?}[F\ res \geq x]$. Quantitative verification returns the maximum probability of eventually reaching a state in M satisfying the property $res \geq x$. Queries for minimum probabilities or those with probability bounds are also easily specified.

To perform cost analysis we augment M with a rewards structure $r_s : S \to \mathbb{R}_{\geq 0}$ defined by $r_s((res, t)) = res$, associating every state (res, t) with the value res. We interpret rewards as the *cost* of the deployment at time t. By using actual resource amounts the customer can scale cost values according to unit resource prices set by the provider. For example, the cost at time t of using Amazon's Standard On-Demand large instances is calculated $res \times 0.34¢$.[2] Using MDPs with costs, the high-level cost analysis query

2. *What is the expected maximum cost of a cloud deployment's resource requirements at any point t?*

is specified by the rewards-augmented PCTL formula $Rmax_{=?}[I = t]$, where $I = t$ denotes the instantaneous cost at time t. Quantitative verification returns the expected maximum cost of M at any point t in the time interval T. Instantaneous costs can be obtained for either single values or any subinterval of T. The high-level cost analysis query

3. *What is the expected maximum cumulative cost of a cloud deployment's resource requirements up to time t?*

is specified by the rewards-augmented PCTL formula $Rmax_{=?}[C \leq t]$, where $C \leq t$ denotes the cumulative reward up to time t. Quantitative verification returns the expected maximum cost of M accumulated over the time interval $[0, t] \subseteq T$.

4 Implementation and Validation

We developed a probabilistic pattern modelling (PPM) tool that implements our approach for the analysis of probabilistic cloud deployment patterns. PPM is an

[2] aws.amazon.com/ec2/pricing (Checked August 2011)

open-source Java class library[3] that supports the realisation of the workflow in Figure 1. The core component of PPM is a `PatternProcessor` class whose constructor takes as parameters the probabilistic pattern (1) to analyse, and the upper bound for the time interval (7). The `PatternProcessor` constructor implements the MDP synthesis technique described in the Section 3.2 by using a parser-generator built using the off-the-shelf language tool ANTLR[4]. The result of this model synthesis is an MDP expressed in the PRISM state-based language.

The public methods of the `PatternProcessor` class (Table 2) enable the quantitative analysis of a range of cost and resource usage properties of the considered probabilistic pattern. Each such method synthesises the appropriate PCTL property as described in Section 3.3, and runs the probabilistic model checker PRISM in the background to analyse this PCTL property against the MDP model generated by the constructor. The result of the PRISM analysis is parsed and returned to the client that invoked the method.

Table 2. Analysis methods provided by the PatternProcessor PPM class

Java	`double getMaxResources(int t)`
	`double getMinResources(int t)`
Input	$t \geq 0$
Output	Expected max(min) resource usage after t hours
PCTL	`Rmax=?[I=t]`, `Rmin=?[I=t]`
Java	`Double[] getMaxResources(int t1, t2, t)`
	`Double[] getMinResources(int t1, t2, t)`
Input	$t_1, t_2 > 0$ such that $t_1 < t_2$ and $0 < t < (t_2 - t_1)$
Output	List of expected max(min) resource usage for each t in $[t_1, t_2]$
PCTL	`Rmax=?[I=t]`, `Rmin=?[I=t]`
Java	`Double[] getMinCumulativeResources(int t1, t2, t)`
	`Double[] getMaxCumulativeResources(int t1, t2, t)`
Input	$t_1, t_2 > 0$ such that $t_1 < t_2$ and $0 < t < (t_2 - t_1)$
Output	Expected max(min) cumulative resource usage over $[t1, t2]$
PCTL	`R=?[C<=t]`
Java	`double getMaxProbResourcesExceeds(int x)`
	`double getMinProbResourcesExceeds(int x)`
Input	$x \geq 0$
Output	max(min) probability of resource usage exceeding x at any time
PCTL	`Pmax=?[F res >= x]`, `Pmin=?[F res >= x]`

To assess the effectiveness and scalability of PPM, we implemented a simple test tool that was used to carry out the case study and scalability experiments presented in the remainder of this section. The results of this validation exercise are being used to improve PPM, with a view to integrate it into an existing high-level tool for cloud adoption decision [13].

[3] PPM is freely available from http://www1.aston.ac.uk/eas/staff/dr-kenneth-johnson/ppm/.

[4] http://www.antlr.org/

4.1 Case Study

The case study described in this section considers a potential cloud customer whose applications require at least three virtual machines at all times in order to maintain a reasonable system response time. This resource requirement can be formalised as a probabilistic pattern with a single baseline declaration:

```
Baseline 3.
```

Each weekday two or three more VMs above the baseline usage are required to be started at 7h. At 9h four or five virtual machines above the baseline usage are required. These requirements are modelled by two rules:

```
Rule1 Start Jan,1,7 Vary {[0.8:bl-add(2) + 0.2:bl-add(3)]}
                    Repeat WeekDay Until Dec,31
Rule2 Start Jan,1,9 Vary {[0.7:bl-add(4) + 0.3:bl-add(5)]}
                    Repeat WeekDay Until Dec,31.
```

Resource usage is reduced at 17h and again at 19h where resources is set back to baseline. These requirements are modelled by two rules:

```
Rule3 Start Jan,1,17 Vary {[0.8:bl-add(2) + 0.2:bl-add(3)]}
                     Repeat WeekDay Until Dec,31
Rule4 Start Jan,1,19 Vary {[1:bl]}
                     Repeat WeekDay Until Dec,31.
```

We used `PPM` to analyse the probabilistic cloud deployment pattern described above. The PRISM MDP model generated as a result of building the PPM `PatternProcessor` object for this pattern is depicted in an abbreviated form in Figure 2.

We display in Figure 3 the results of analysing the pattern's maximum expected resource usage over the time interval ($72 \leq t \leq 96$) representing January 3^{rd}, a single weekday. The figure labels each application of a rule by the rule name. As the pattern indicates, the number of VMs at the beginning of the day is 3 and peaks between the hours of 9am and 5pm (when $79 \leq t \leq 90$) when the expected maximum resource usage has value 7.3. The resource usage returns to the baseline outside working hours at 7pm (when $t \geq 91$).

Figure 4 depicts the results of cost analysis performed to determine monthly maximum expected accumulated costs from January to April. The four points on the graph are labeled with the cloud resource usage and the end of each month. These values can be used to determine the maximum expenditures for cloud computing services expected at the end of a provider's billing period. For example, supposing the customer uses Amazon's standard EC2 instance at a unit cost of 0.32¢ throughout January, the maximum expected expenditure at the end of January is $3139.2 \times 0.32¢ = \$1004.55$.

4.2 Scalability

Experiments were performed on the `PPM` tool to test the scalability of our approach using a set of different patterns of increasing complexity defined over

```
mdp
...
const BASELINE=3;
formula Rule1  = (m=Jan)&(d=3)&(h=6);
const double p1=0.8; const double p2=0.2;
const Amount1 = 2; const Amount2 = 3;
formula Vary1 = min(BASELINE+Amount1,MAXRES);
formula Vary2 = min(BASELINE+Amount2,MAXRES);
...
formula RuleStart = (Rule1| ... );

rewards true : res; endrewards
module pattern
   m : [0..11] init Jan; d : [1..31] init 1; h : [0..23] init 0;
   res: [0..20] init BASELINE;

[] (!DayEnd)&(!RuleStart) -> (h'=h+1);
[] (!DayEnd)&(Rule1) -> p1:(res'=Vary1)&(h'=h+1)+
                        p2:(res'=Vary2)&(h'=h+1);
...
[] (DayEnd)&(NotMonthEnd)&(!RuleStart) -> (h'=0)&(d'=d+1);
[] (DayEnd)&(NotMonthEnd)&(Rule1) -> p1:(res'=Vary1)&(h'=0)&(d'=d+1)+
                                     p2:(res'=Vary2)&(h'=0)&(d'=d+1);
...
[] (DayEnd)&(MonthEnd)&(m<Dec)&(!RuleStart) -> (m'=m+1)&(h'=0)&(d'=1);
[] (DayEnd)&(MonthEnd)&(m<Dec)&(Rule1)  ->
     p1:(res'=Variation1)&(m'=m+1)&(h'=0)&(d'=1) +
     p2:(res'=Variation2)&(m'=m+1)&(h'=0)&(d'=1);
...
[] (m=Dec)&(DecEnd)&(DayEnd)-> true;
endmodule
```

Fig. 2. Abbreviated PRISM file generated by case study analysis

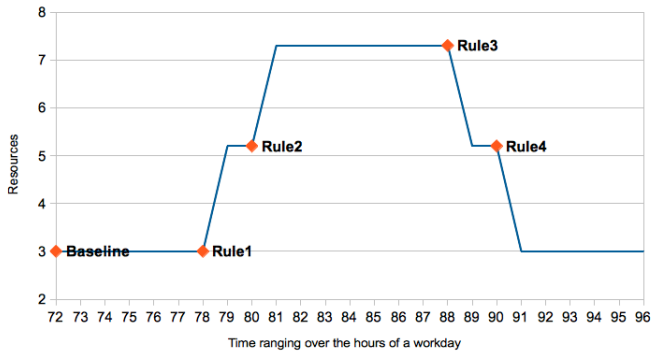

Fig. 3. Expected maximum resource usage for January 3^{rd} (hours 72 to 96).

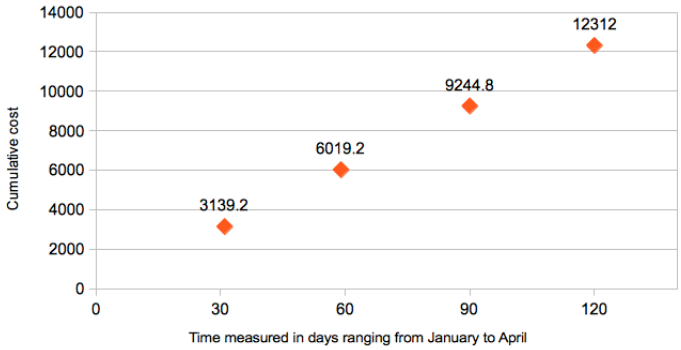

Fig. 4. End of month expected accumulated costs from January to April

a single week beginning January 1^{st}. Rules formed with the `Repeat` construct with a frequency of `WeekDay` increased resource amounts and were applied incrementally on successive days in the week ranging from one to five. Rules started from 0h staggered by four hours and one or two probabilistic distributions were specified in each rule.

The experiments recorded the number of states and transitions for each MDP model synthesised from the patterns. The analysis speed of `PPM` was tested by performing two method invocations from the Java library on each pattern:

- `getMaxResources(0,168,1)`, calculating the maximum expected resource usage for each hour of the week, and
- `getMaxCumulativeResources(0,168,1)`, calculating the maximum expected cumulative cost over each hour of the week.

Table 3. Model size and method invocation time according to pattern complexity

Pattern size		Model size		Avg. analysis time (minutes)	
Rules	Dists	States	Transitions	Max. Usage	Max. Cost
1	1	413	423	1.96	2.98
2	1	758	803	4.81	4.29
3	1	1083	1188	5.57	3.48
4	1	1279	1450	5.39	8.08
5	1	1378	1609	10.91	9.14
1	2	901	967	5.65	6.18
2	2	1936	2287	14.58	13.68
3	2	2911	3772	21.69	22.5
4	2	3499	4930	25.69	25.26
5	2	3796	5749	33.49	30.98

Our experiments were performed using the `PPM` command line interface on an Apple MacBook Pro operating on Mac OS X Version 10.6.8 with a 2.66Ghz Intel Core 2 Duo processor and 8GB of 1067MHz DDR3 memory. PRISM version 4.0.beta was used to perform the quantitative analysis on the synthesised graphs,

and the sparse matrix engine appeared to produce the best running times. The results of the experiments are listed in Table 3 where method invocation times are average over several runs, and include all steps in processing the pattern: parsing, model synthesis and verification.

Fig. 5. Scalability results: state space according to pattern complexity

Figure 5 shows the growth in state space complexity according to the number of rules. To analyse the size complexity of the state space S of an MDP modelling a pattern, we introduce the following notation. The *size* of a rule R with m probability distributions each with q_i probabilities, $1 \leq i \leq m$ is defined as $size(R) = q_1 + \cdots + q_m$. We assume that the maximum rule applications that can be made at any single time t is bounded by the value K. For a pattern with rules $R_1, \ldots, R_r$ we set this value to satisfy the inequality $size(R_1) + \cdots + size(R_r) \leq K$. We adapt Equation 8 to give the inequality

$$|S^{t+1}| \leq \begin{cases} |S^t| \cdot K & \text{if a rule(s) in } P \text{ starts at time } t \\ |S^t| & \text{otherwise,} \end{cases} \tag{10}$$

for all $t \geq 1$ and $|S^0| = 1$. Using Inequality 10, and noting the inequality $|S| \leq |S^0| + |S^1| + |S^2| \cdots + |S^n|$ we calculate a bound on the size $|S|$ of the state space over the interval $[0, n]$. For patterns consisting of only a baseline declaration, we have $n + 1 \leq |S|$, e.g. a single state models each point in the time interval. Now, using the `Repeat` construct, rules can be applied repeated on a given frequency which we denote by g and this yields $|S| \leq \sum_{t=0}^{n} K^{\lceil \frac{t}{g} \rceil}$. In the worst-case scenario, when rules are applied at each time in the interval with frequency $g = 1$, we have

$$|S| \leq \frac{K^{n+1} - 1}{K - 1}.$$

5 Related Work

There has been significant research in cloud computing focused on developing tools and techniques to assist organisations in assessing the cost-savings of tran-

sitioning to the cloud. Specific aspects of cloud technology such as Infrastructure as a Service (IaaS) have been formally modelled to analyse cost-savings of leasing CPU[21] and data storage[22] from a remote data centre, while case studies assessing feasibility of cloud computing has been carried out for specific industries[14] and applications[7].

Advanced tools such as CloudSim [2] model components of cloud computing data centres for fine-tuning applications deployed on the cloud. The tool enables users to improve application performance by simulating resource provisioning policies, and work is in progress to extend support to include simulated costing-analysis of deployment on public clouds [3].

Research undertaken as part of the LSCITS[5] initiative has developed the Cloud Adoption Toolkit [13] which is an organisational framework identifying key concerns of cloud services adoption with tools to support customer's decision making process. In particular the framework's cost modelling tool allows cost-analysis of cloud deployments to be performed with resource requirements expressed in a notation similar to the language developed in our approach.

Our work compliments and improves upon these approaches by accounting for probabilistic behaviour of cloud deployments and data centres and using quantitative verification techniques for costing and resource usage analysis.

6 Conclusion and Future Work

The results presented in this paper target the growing need for precise cost analysis techniques that address both uncertainty and probability in using cloud computing services. The approach introduced in the paper formalises resources used by a cloud computing deployment as a probabilistic pattern. Markov decision process are synthesised and quantitative analysis applied to provide the customer with accurate costing and usage results. The approach has been implemented as an open-source Java package called `PPM` which uses the probabilistic model checker PRISM to perform verification on MDPs synthesised from patterns. We have validated our approach with a realistic case study and scalability experiments.

Our future work aims at improving the runtime performance and usability of the `PPM` tool. We envision a number of directions including capabilities of background pre computation of quantitative results for later use, and extending capability to enable performing multiple cost analysis functions concurrently (via Java threads). We can also improve scalability performance by exploiting the periodical nature of some patterns. For example, the case study in Section 4.1, applies the same pattern rules to each workday). Lastly, we envision integration of the tool to existing toolkits such as [13] and develop software components to derive probabilistic patterns from observed resource requests frequencies in application logs.

[5] http://lscits.cs.bris.ac.uk/

7 Acknowledgements

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

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A Berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, Electrical Engineering and Computer Sciences University of California at Berkeley (2009)
2. Buyya, R., Ranjan, R., Calheiros, R.: Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In: High Performance Computing Simulation, 2009. HPCS '09. International Conference on. pp. 1 –11 (june 2009)
3. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience 41(1), 23–50 (2011)
4. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS management and optimization in service-based systems. IEEE Transactions on Software Engineering 37, 387–409 (2011)
5. Calinescu, R., Johnson, K., Rafiq, Y.: Using observation ageing to improve Markovian model learning in QoS engineering. In: Proceeding of the second joint WOSP/SIPEW international conference on Performance engineering. pp. 505–510. ICPE '11, ACM, New York, NY, USA (2011)
6. Calinescu, R., Kikuchi, S.: Formal methods @ runtime. In: Calinescu, R., Jackson, E. (eds.) Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems, Lecture Notes in Computer Science, vol. 6662, pp. 122–135. Springer Berlin / Heidelberg (2011)
7. Deelman, E., Singh, G., Livny, M., Berriman, B., Good, J.: The cost of doing science on the cloud: the montage example. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. pp. 50:1–50:12. SC '08, IEEE Press, Piscataway, NJ, USA (2008)
8. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6, 512–535 (1994)
9. Jeannet, B., D'Argenio, P., Larsen, K.: Rapture: A tool for verifying Markov decision processes. In: Cerna, I. (ed.) Proc. Tools Day, affiliated to 13th Int. Conf. Concurrency Theory (CONCUR'02). pp. 84–98. Technical Report FIMU-RS-2002-05, Faculty of Informatics, Masaryk University (2002)
10. Jensen, M., Schwenk, J., Gruschka, N., Lo Iacono, L.: On technical security issues in cloud computing. Cloud Computing, IEEE International Conference on 0, 109–116 (2009)
11. Joint, A., Baker, E., Eccles, E.: Hey, you, get off of that cloud? Computer Law & Security Review 25(3), 270 – 274 (2009)
12. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Performance Evaluation 68(2), 90 – 104 (2011)

13. Khajeh-Hosseini, A., Greenwood, D., Smith, J.W., Sommerville, I.: The cloud adoption toolkit: supporting cloud adoption decisions in the enterprise. Software: Practice and Experience (2011)
14. Khajeh-Hosseini, A., Greenwood, D., Sommerville, I.: Cloud migration: A case study of migrating an enterprise IT system to IaaS. Cloud Computing, IEEE International Conference on 0, 450–457 (2010)
15. Kikuchi, S., Matsumoto, Y.: Performance modeling of concurrent live migration operations in cloud computing systems using PRISM probabilistic model checker. In: Proc. 4th International Conference on Cloud Computing (IEEE Cloud 2011) (2011)
16. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11). pp. 585–591. Springer (2011)
17. Kwiatkowska, M., Parker, D., Qu, H.: Incremental quantitative verification for Markov decision processes. In: Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on. pp. 359 –370 (june 2011)
18. Kwiatkowska, M.: Quantitative verification: models techniques and tools. In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. pp. 449–458. ACM, New York, NY, USA (2007)
19. Rutten, J., Kwiatkowska, M., Norman, G., Parker, D.: Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems, P. Panangaden and F. van Breugel (eds.), CRM Monograph Series, vol. 23. American Mathematical Society (2004)
20. 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 (December 2008)
21. Walker, E.: The real cost of a CPU hour. Computer 42(4), 35 –41 (april 2009)
22. Walker, E., Brisken, W., Romney, J.: To lease or not to lease from storage clouds. Computer 43(4), 44 –50 (april 2010)
23. Youseff, L., Butrico, M., Da Silva, D.: Toward a unified ontology of cloud computing. In: Grid Computing Environments Workshop, 2008. GCE '08. pp. 1 –10 (nov 2008)