Multi-objective Optimisation for Rolling Upgrade Allowing for Failures in Clouds

Daniel Sun*‡, Daniel Guimarans†, Alan Fekete*§, Vincent Gramoli*§, and Liming Zhu*‡§
*Software System Research Group, NICTA, Australia
†Optimization Research Group, NICTA, Australia
‡The University of New South Wales, Australia
§University of Sydney, Australia
Email: {daniel.sun, alan.fekete, daniel.guimarans, vincent.gramoli, liming.zhu}@nicta.com.au

***Abstract*—Rolling upgrade is a practical industry technique for online updating of software in distributed systems. This paper focuses on rolling upgrade of software versions in virtual machine instances on cloud computing platforms, when various failures may occur. An operator can choose the number of instances that are updated in one round and system environments to minimise completion time, availability degradation, and monetary cost for entire rolling upgrade, and hence this is a multi-objective optimisation problem. To predict completion time in the presence of failures, we offer a stochastic model that represents the dynamics of rolling upgrade. To reduce the computational effort of decision making for large scale complex systems, we propose a technique that can find a Pareto set quickly via an upper bound of the expected completion time. Then an optimum of the original problem can be chosen from this set of potential solutions. We validate our approach to minimise the objectives, through both experiments in Amazon Web Service (AWS) and simulations.**

I. Introduction

Software upgrade in enterprise systems, especially in large scale web service systems, is inevitable and its frequency has been increasing. The research challenges of doing this in distributed systems that must remain highly available, while handling failures, have been recognised a decade ago [1]; The vulnerability of distributed software upgrade has been explored [2], [3]. Currently, the common practice in industry is to use rolling upgrade [3]: A small number of instances are changed in one round, from the old version (V_1) to the new one (V_2), and then another round changes another small set of instances. This process is repeated in a wave rolling throughout a system, until all instances are of V_2. This paper focuses on rolling upgrade of a software application deployed in clouds, where it is economical and simple to update software by thoroughly replacing some virtual machine instances in V_1 with newly created ones in V_2. The advantage of rolling upgrade is that availability does not degrade too much.

Both previous studies and industrial experiences have revealed that rolling upgrade is vulnerable to diverse failures. During a rolling upgrade, there are many sources of failures such as system hardware, software infrastructure, the virtualisation layers, and even applications in a cloud. All of potential failures can significantly affect the upgrade process. Thanks to virtualisation technology, failures can be efficiently tolerated by replacing and migrating failed virtual machines. However, actively replacing failed instances brings uncertainty to the progress of rolling upgrade: For a software system running in clouds, there are a bundle of backup images. In clouds modifying images of running virtual machines is often warned [18], [19]. A software version of the backup images should be stable and reliable, and consequently is too critical to be replaced before its next version has been well tested and confirmed from online experience. That is, any newly created instances must be in V_1. In a rolling upgrade, while some instances are being upgraded from V_1 to V_2, some of V_2 may be returning to V_1 due to failures.

In practice on planning a rolling upgrade in a cloud-hosted application, an operator would like to achieve some goals: completing the process quickly, paying as little monetary cost as possible, and keeping high service availability. The operator has some choices to trade off, the number of instances to upgrade at one round, system environments, and operation types: The more instances are being upgraded in each round, the less instances can provide service, but the earlier the rolling upgrade can be finished. Failures can degrade availability and extend completion time, but an operator can prepare for a rolling upgrade by adjusting system environments to reduce software, system, and performance interference, by moving partial workload off the instances, by employing more spare instances and replicating services, and by adopting more reliable upgrade operations. The preparation can lead to more stable and healthier system environments which imply low failure probabilities, but monetary cost applies.

In this paper, our first contribution is the formulation of multi-objective optimisation for simultaneously minimising the completion time (T), the cost (C), and the expected loss (L) of service instances with the presence of failures. We name this optimisation to be the *TCL problem*. To solve the TCL problem, it is necessary to understand system dynamics during rolling upgrade, and to this end we offer the second contribution: a stochastic model that represents rolling upgrade processes for computing T. With the model the TCL problem is theoretically solvable, but it is computationally expensive for large-scale complex systems. Hence, our third contribution is a technique: we use an upper bound of the expected completion time to find a Pareto set, in which we search for the minimum of scalarized objective function, given the weights chosen by an operator. Our evaluations show that our model gives good predictions of the metrics, and our search-reduction optimisation finds the TCL solutions in much reduced computational effort, while for some weights the solution we find is sub-optimal but still close to the global optimum.

The rest of this paper is structured as follows. In Section II, we review the related work. In Section III, we present the assumptions and the terminology in this paper. In Section IV, the TCL optimisation problem is formulated. In Section V we use absorbing Markov chains to provide the computation of predicted completion time. In Section VI we use an upper bound of expected completion time to solve the problem efficiently. In Section VII, our experiments are described, and the results are presented. Section VIII concludes this paper and reflects our contributions.

II. Related Work

On cloud platforms, upgrading instances can be carried out by simply shutting down and restarting virtual machines [4]. In [3], knowledge of faults is assumed and a fault model is proposed to help a system. A probabilistic risk model has been proposed to understand the risk by comparing different factors in [5]. In this way, it is possible to decide whether to upgrade or not, but not possible to control and predict rolling upgrade in runtime. In [6], [7], understanding the dynamics of software updating with the presence of failures has been recognised to be essential for distributed software updates. Software, performance, and fault interferences exist in software systems. Interferences can have heavy impact on failures and faults. In clouds interference among tenants and virtual machines is also considerable. It is possible to moderate system environments to reduce faults and improve performance [8]–[11].

Reliability and availability of software and hardware systems have been well analysed with stochastic models [12]–[15]. Recently we have reported on risk quantification for version switch in rolling upgrade [15] and a distributed rolling upgrade protocol [16], both of which represent system dynamics with absorbing Markov chains under different assumptions on failures. The computation time of numeric models may be considerably long and this hampers the practical usage in many cases. In [17], a technique is addressed for deriving an upper bound of expected completion time for specific absorbing Markov chains with little computation effort.

To implement our approach, we need AutoScaling, which aims at elasticity and has been a standard module in popular cloud software stacks [18], [19]. We also need information retrieval and extraction from software systems. The techniques have been well surveyed [13]. There are also some techniques and tools for exactly acquiring failure probabilities. In the context of data centres, the failure probabilities can be estimated as the ratio of the number of components of a given type that failed during a time period over the total number of components of this type [20]. Regarding the failure probabilities of software dependencies, the Common Vulnerability Scoring System (CVSS) [21] can be used to provide vulnerability-related failure probabilities for many software libraries and packages. In order to deal with software faults and errors, our previous work on process context mining and log analysis, detection and diagnosis of software flaws and configuration errors, and risk control in [15], [22] are also necessary for a full-fledged implementation.

III. Preliminaries

In this section, we present the formalisation and necessary notations. For clarity of language, in the following we say *an operation* for an attempt to replace a single instance in V_1 with a new instance in V_2, *a round* for the simultaneous operations that replace a small number of the instances, and *rolling upgrade* or *process* for the entire activity (in multiple rounds) that upgrades all the N instances. The notations we are using are listed in Table I.

TABLE I. Notations

Symbols	Descriptions
N	Number of all instances
G	Granularity, the number of instances in one upgrade
p_p	Probability of platform failure for an individual instance
p_o	Probability of an individual operation failure
$\mathbb{K}$	Set of system options, i.e. (p_p, p_o) pairs
T or $T(p_p, p_o, G)$	Expected time (in rounds) to complete rolling upgrade
$B(p_p, p_o, G)$	Bound of expected completion time $T(p_p, p_o, G)$
C or $C(p_p, p_o)$	Financial cost of a given option
$\mathbb{C}$	Set of costs among different options
L or $L(p_p, G)$	Expected loss of service instances per round
U	Number of upgraded instances
g	Number of failures in instances being upgraded
u	Number of failures in upgraded instances
v	Number of operation failures

A. System Model

The system consists of a set of instances that initially are ready to be upgraded. A rolling upgrade is performed in consecutive rounds, and in each round a subset of V_1 instances are selected. The size of the subset, i.e. the *granularity* value G, can be chosen by the operator. Then, G operations attempt to replace the selected instances simultaneously in one round. After all new instances have been tested, a round is finished and the next one can start. We measure the completion time of the entire rolling upgrade process in the number of all rounds, which are assigned the same timeout. In each round an individual instance is given only one attempt, and if it is not successful, another attempt to the same instance will be in a later round.

The progress of the rolling upgrade is characterised by the number of V_1 instances. We refer to the situation that exactly i instances are healthily running V_1 (and thus there are $N - i$ instances running V_2), as state i of the rolling upgrade[1]. When i reduces from N to 0, we say that a rolling upgrade has been successfully finished. The assumptions necessary for modelling and formulating the problem are listed as follows.

- $\mathcal{A}_1$ Software and configuration faults can be detected, and failures will be handled by a recovery.
- $\mathcal{A}_2$ An instance can only fail once in a round. This makes the model simpler, and is a reasonable approximation to reality because a round is not much longer than the replacement time.
- $\mathcal{A}_3$ The operator can obtain information such as failure probabilities or rates for each system option through retrieving or extracting systems information.
- $\mathcal{A}_4$ The operator is aware of the possible system options and can obtain the information about the corresponding costs.

Note that, one may argue that a rolling upgrade can be scheduled to an idle time slot in a system, but we do not consider

[1]This descending numbering makes for analytical convenience later

this for two reasons: First a rolling upgrade may take a long time, hours or days, in large scale systems and applications such that there is no time slot long enough; Second for those systems that do not provide continuous service and indeed have sufficient idle time, rolling upgrade is not necessary.

B. Failure and Cost

Since we use instance health checking and auto-scaling provided by cloud infrastructures to deal with instance and infrastructure failures, we only need to consider the features of those failures other than what they are and how they can be tolerated. This is the benefit from cloud technologies. We name this kind of failures *Platform failures*, which randomly appear in any instances. Since all rounds are assigned the same timeout, the probability, p_p, of platform failures for an individual instance in a round can be computed from the operator's knowledge (see $\mathcal{A}_3$). Thus, the number of platform failures during a round follows a binomial distribution characterised by p_p. An attempt to upgrade an instance may fail, sometimes frequently. We call this kind of failures *Operation failures*. From sufficient software test, the number of operation failures can be known and it also follows a binomial distribution with the probability p_o. For all the other software and configuration faults, we have assumed $\mathcal{A}_1$ and will not consider in this paper.

The recovery of failures does not impact the process, since the recovered instances are in V_1. If all instances are in V_2 except some instances not recovered yet in the final rounds, the operations will directly create new V_2 instances.

The operator has a number of *system options* (see $\mathcal{A}_4$). Each option corresponds to particular system environment and operation, which are characterised by a pair of (p_p, p_o). We regard p_p and p_o as discrete parameters. The pairs of $K_i = (p_p, p_o)$ are contained into a set $\mathbb{K}$. Thus, K_i represents a system option for rolling upgrade. Each system option is associated with a corresponding financial cost. The cost data are available (see $\mathcal{A}_4$). The costs are in the set $\mathbb{C}$. There is a mapping between $\mathbb{K}$ and $\mathbb{C}$, $C : \mathbb{K} \mapsto \mathbb{C}$. Because different environments may result in the same cost, $|\mathbb{K}| \geq |\mathbb{C}|$. To know the exact form of C is difficult and there is no existing cost functions for cloud context in the literature. The cost data can be collected from specific systems, calculated from the price list of pay-as-you-go if available, or extracted from maintenance cost.

IV. The TCL optimisation problem

An operator can decide on a granularity G, and choose a system option with particular failure probabilities, for the tradeoff among the goals. The expected completion time depends on p_p, p_o, and G and consequently is denoted as a computation $T(p_p, p_o, G)$; The cost is known as $C(p_p, p_o)$; The expected loss of service instances can be calculated as follows: In each round, some of the instances of the service are not available to respond to clients, either because those instances are being operated, or because they are being recovered after failures. Since the recovery is actually to replace failed instances, the recovery time can be approximately the time length of a round. It is not difficult to see that L depends also on p_p, and then we derive the expected number of lost instances per round,

$$L(p_p, G) \quad = \quad (N - G)p_p + G. \tag{1}$$

The tradeoff is that if an operator chooses a system option with smaller failure probabilities for shorter completion time and less expected loss of instances, the cost will be higher, while a greater G means fewer rounds but more loss of instances in each round. Generally, an operator needs to minimise T, C and L. Then, we formulate the multi-objective optimisation problem, i.e. the TCL problem.

$$\begin{array}{ll} \min & \{T(p_p, p_o, G), C(p_p, p_o), L(p_p, G)\} \\ s.t. & \{p_p, p_o\} \in \mathbb{K} \\ & 1 \leq G \leq N \end{array} \tag{2}$$

The TCL problem can be solved in many ways [23]. In this paper, we adopt the scalarisation technique, by which the TCL problem is in the following form.

$$\begin{array}{ll} \min & f = \alpha T(p_p, p_o, G) + \beta C(p_p, p_o) + \gamma L(p_p, G) \\ s.t. & \{p_p, p_o\} \in \mathbb{K} \\ & 1 \leq G \leq N, \end{array} \tag{3}$$

and $\alpha + \beta + \gamma = 1$. The benefit of adopting the scalarization technique is that the weights α, β, and γ can be set by an operator to give appropriate emphasis on the objectives to reflect the importance of a particular setting. Note that, in some cases magnitude differences may have obvious impact on the result. It is not difficult to adopt a transformation method to remove the differences, about which in this paper we do not discuss.

In the TCL problem, G, p_p, and p_o are all discrete; $C(p_p, p_o)$ is known; $L(p_p, G)$ is in a closed form. Thus the essential step is to compute an accurate $T(p_p, p_o, G)$. In the next section, we show the computation through stochastic modelling.

V. Modelling Rolling Upgrade with DTMC

In this section, we provide the computation $T(p_p, p_o, G)$ for predicting expected completion time. The nature of rolling upgrade consisting of consecutive rounds implies a discrete time model, and therefore we choose discrete time Markov chain (DTMC) to represent the dynamics of rolling upgrade. Since a process ends at all instances being in V_2, there must be an absorbing state. We use the system states as the state space of the absorbing Markov model, $\{0, 1, 2, \cdots, N\}$, where 0 denotes that there is no instance remaining in V_1. A process starts from N and once it reaches the state 0, it is successfully finished. In each round, the state moves forward or backward, or stays in the same state as before. The farthest forward transition distance is G. In other words, if the current state is i the next state after a forward transition is at most $i - G$ (no failure presents). On the other hand, the next state after a backward transition from i can be N(there are too many failures). Some examples with different G are shown; in Fig. 1. In this paper, we provide below a general solution for all $1 \leq G \leq N$.

We need to calculate the transition probabilities in terms of the probabilities of failures. For a forward transition from the state i to $i - j$, the number of successful operations should be j at least. The number of successful operations can be greater

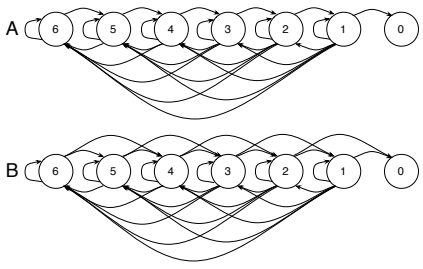

Fig. 1. Examples of chains A ($N = 6$, $G = 1$) and B ($N = 6$, $G = 2$).

than j, when some V_2 instances are lost due to failures. Since in every round, a rolling upgrade operates on G instances, it is necessary to understand what have happened to the other $G-j$ instances. If g instances in G fail, the number of operations is actually $G - g$, but $G - g \geq j$ because at least j successful operations must be performed for the transition from i to $i-j$. Hence, we know $g \leq G - j$. Then if $G - j - g > 0$, there must be u platform failures in V_2 instances and v operation failures, and otherwise the state will move farther than j steps. For this reason, $u + v = G - j - g$ must hold. In other words, the number of actual operations should be $u + v + j = G - g$, among which the number of successful operations should be $u + j$ since v operations must fail. Note that, the failures in V_1 instances have no impact on any transitions. By summing up, the forward transition probability is (4).

Since $U = N - G - i$, the probability is a function of i. Similarly for the backward transition probability, the net loss of V_2 instances should be j. Hence, the number of failures in V_2 instances should be $u \geq j$ and $u \leq G + j$ since only failures can move the states backward. The number of successful operations should be $u - j$ out of $G - g$ operations and as a result the other $G - g - u + j$ operations must be with operation failures. The backward transition probability is (5).

$$p_{i,i-j} = \sum_{g=0}^{G-j} \sum_{u=0}^{G-j-g} \left(\binom{G}{g} \binom{U}{u} p_p^{g+u} q_p^{G+U-g-u} \binom{G-g}{u+j} p_o^{G-j-g-u} q_o^{u+j} \right). \quad (4)$$

$$p_{i,i+j} = \sum_{g=0}^{G} \sum_{u=j}^{G+j-g} \left(\binom{G}{g} \binom{U}{u} p_p^{g+u} q_p^{G+U-g-u} \binom{G-g}{u-j} p_o^{G-g-u+j} q_o^{u-j} \right). \quad (5)$$

It is possible that after an upgrade the state does not change at all. This probability is $p_{i,i}$ in (6), since we only consider the case that all $p_{i,j}$ add up to 1.

$$p_{i,i} = 1 - \sum_{j=1}^{N-i} p_{i,i+j} + \sum_{j=1}^{G} p_{i,i-j}. \quad (6)$$

Note that in the modelling G should be replaced by i iff. $i < G$ because at the final rounds the granularity is possible to be smaller than G. Because the V_1 instances are not involved in the transition probabilities, the V_1 instances do not impact the accuracy of our modelling.

With all the transition probabilities, the stochastic matrix $\mathbf{P}$ can be established. Then, it is not difficult to compute the expected completion time in terms of [12]. In this paper, we do not repeat the content in textbook. As a result, we can compute $T(p_p, p_o, G)$ from the model. As we will see in Section VII, the accuracy of $T(p_p, p_o, G)$ is satisfying. Thus, the TCL problem becomes solvable.

VI. Solving TCL Problem Efficiently

Because both $T(p_p, p_o, G)$ and $C(p_p, p_o)$ are not explicit functions, we need a search algorithm to find the optima, but it is inevitable to query the model for $T(p_p, p_o, G)$ many times. Each query has to compute an inverse of a matrix with considerable computation effort. In larger-scale systems, the search space may be huge and consequently the transition matrix is big. Therefore large-scale optimisations become difficult. If we can compare completion times from different parameters and shrink the search space without matrix computation, the optimisation can be solved more efficiently. That implies such a computation $B(p_p, p_o, G)$: It must be computed quickly; If $T(p_p^1, p_o^1, G^1) \leq T(p_p^2, p_o^2, G^2)$, we always have $B(p_p^1, p_o^1, G^1) \leq B(p_p^2, p_o^2, G^2)$. In this paper, we adopt an upper bound of the expected completion time and use this bound as $B(p_p, p_o, G)$. Then we evaluate the effectiveness and the efficiency through experiments.

A. An Upper Bound of Completion Time

At first, let us discuss the model in a formal way. The model has a state space $\mathbf{\Pi} = \{0, 1, 2, 3, \cdots, N\}$. A chain is a series of random variables $\{\pi_j; j \in \mathbf{N}\}$. The absorbing state is 0 and a process starts from $\pi_0 = N$. Let T_N be the completion time from the initial state N. Then, we introduce the upper bound of $E[T_N]$ through the drift, which is defined to be $\Delta_i = \sum_{l=1}^{N-i} l p_{i,i+l} - \sum_{l=1}^{G} l p_{i,i-l}$. Given the granularity and the probabilities, Δ_i keeps monotonicity as shown in Lemma 1 and 2. Because the space limit, we do not derive the bound in this paper. The proofs can be reached in [17].

Lemma 1. *For a transition matrix generated by (4), (5), (6), as i decreases, i.e. a rolling upgrade progressing, Δ_i keeps increasing always.*

Lemma 2. *If $\Delta_1 > 0$, there must be a state τ such that $\forall i < \tau$, $\Delta_i < 0$ and $\forall i \geq \tau$, $\Delta_i \geq 0$.*

In practice, computing our bounds only needs the monotonicity of Δ_i, and actually it does not need to construct the matrix and compute all Δ_i. For the case that both positive and negative drifts exist, the existence of τ splits the entire state space into two parts: One is the states with negative drifts and the other is the states with positive drifts. Let the latter be a sub-space $\mathbf{\Omega} = \{i \in \mathbf{\Pi}, \Delta_i \geq 0\}$, in which all states are with non-negative drifts, and in $\mathbf{\Pi} - \mathbf{\Omega}$ all states are with negative drifts. Physically this implies that such a rolling upgrade proceeds quickly before τ and slows down until the finish of the whole rolling upgrade. The upper bound of expected completion time for $\mathbf{\Omega} = \varnothing$ is in Theorem 3.

Theorem 3. *If* $\mathbf{\Omega} = \varnothing$, $E[T_N] < \lceil \frac{\lg N}{\lg \mu} \rceil + \frac{\mu}{\mu - 1}$, *where* $\mu = \frac{N}{N + \Delta_1}$.

When $\mathbf{\Omega} \neq \varnothing$, we provide Theorem 4 to solve this problem, but we only provide the case of $0 < \Delta_1 < 1$, since when $\Delta_1 \geq 1$ the completion time must be unreasonably long with respect to the number of instances. In this paper, we filter off those cases resulting in $\Delta_1 \geq 1$ immediately.

Theorem 4. *For* $\mathbf{\Omega} \neq \varnothing$, $0 < \Delta_1 < 1$ *and* $\Delta_\tau \geq 0$, *let* $T_\tau^{\mathbf{\Omega}}$ *represent the time spent in* $\mathbf{\Omega}$ *before returning back to* $\mathbf{\Pi} - \mathbf{\Omega}$ *or falling into the final absorbing state, and let* $T_{N-\tau}^{\mathbf{\Pi}-\mathbf{\Omega}}$ *be the time before entering* $\mathbf{\Omega}$, *the expected completion time of a rolling upgrade is bounded as*

$$E[T_N] \quad \leq \quad E[C_{\mathbf{\Omega}}]\big(E[T_\tau^{\mathbf{\Omega}}] + E[T_{N-\tau}^{\mathbf{\Pi}-\mathbf{\Omega}}]\big),$$

where $C_{\mathbf{\Omega}}$ *is the number of commutes between* $\mathbf{\Omega}$ *and* $\mathbf{\Pi} - \mathbf{\Omega}$, *and*

$$E[T_{N-\tau}^{\mathbf{\Pi}-\mathbf{\Omega}}] \quad < \quad \left\lceil \frac{\lg(N-\tau)}{\lg \mu} \right\rceil + \frac{\mu}{\mu - 1}, \mu = \frac{N-\tau}{i + \Delta_{N-\tau}},$$

$$E[T_\tau^{\mathbf{\Omega}}] \quad = \quad \frac{1}{\Delta_1} + \frac{(\tau+1)(E[C_{\mathbf{\Omega}}])^{\tau+1}}{(E[C_{\mathbf{\Omega}}])^{\tau+1} - 1} \cdot \frac{2}{1 - \Delta_1},$$

$$E[C_{\mathbf{\Omega}}] \quad = \quad \frac{1 - \Delta_1}{1 + \Delta_1}.$$

As a result, $B(G, p_p, p_o)$ is the bound in either Theorem 3 or Theorem 4.

B. Finding Pareto Sets and Proposed Optimisation Method

With the aforementioned bound, the optimisation in (2) can be solved in two steps. First, we search for a Pareto set $\mathbb{S}^*$, i.e. Pareto surface, for the following optimisation problem.

$$\begin{array}{ll} \min & \{B(p_p, p_o, G), C(p_p, p_o), L(p_p, G)\} \\ s.t. & \{p_p, p_o\} \in \mathbb{K} \\ & 1 \leq G \leq N \end{array} \tag{7}$$

$\mathbb{S}^* = \{p_p^*, p_o^*, G^* | \{p_p, p_o, G | B(p_p^*, p_o^*, G^*) \leq B(p_p, p_o, G), C(p_p^*, p_o^*) \leq C(p_p, p_o), L(p_p^*, G^*) \leq L(p_p, G), \forall p_p, p_o, G\} = \varnothing\}$. Second, we search only over $\mathbb{S}^*$ for an optimum in terms of the following scalarisation.

$$\begin{array}{ll} \min & f = \alpha T(p_p, p_o, G) + \beta C(p_p, p_o) + \gamma L(p_p, G) \\ s.t. & \{p_p, p_o, G\} \in \mathbb{S}^*, \end{array} \tag{8}$$

and $\alpha + \beta + \gamma = 1$. It has been proved that the minimum produced by (3) is an efficient solution for the original multi-objective optimisation problem and belongs to the original Pareto set $\mathbb{S}$ of (2). If $B(p_p, p_o, G)$ can distinguish and compare different solutions well, $\mathbb{S}^*$ is identical or close to $\mathbb{S}$, and hence the minimum from (8) is the same as or close to (3). Because any change to the parameters results in different Δ_i, which impact on the bounds heavily, the bound can trace the change of the expected completion time. In this paper, we evaluate the effect in experiments. Both $\mathbb{S}$ and $\mathbb{S}^*$ must be smaller than the overall search space.

Now, actually we have two optimisation methods: $\mathcal{M}_1$ is our proposed method that finds $\mathbb{S}^*$ via the bound for (7) and then searches for the solutions in $\mathbb{S}^*$ for minimising the objective function in (8); $\mathcal{M}_2$ is to find $\mathbb{S}$ directly via the model and then searches for the solutions in $\mathbb{S}^*$. It is possible to search for an optimum directly from the search space, but we do not discuss this possibility in this paper, because a notable benefit from $\mathcal{M}_1$ and $\mathcal{M}_2$ is that once a Pareto set is found whatever α, β, and γ are from users, the optimisation can be solved on the set because all optima must be in the Pareto set. Also we omit the analytical comparison between the two methods, since $\mathcal{M}_1$ is obviously faster than $\mathcal{M}_2$ for no matrix operations involved.

VII. Empirical Studies

In this section we examine the validity of our model, and the proposed optimisation method. We have some experiments done in AWS, among instances which work in a cooperative mode to host distributed HTTP servers. In our AWS experiments, we have artificially introduced failures by killing instances with appropriate probabilities. In AWS, we examine only settings where $N = 32$, as these experiments cost real money. We also have run discrete event simulations, in which we watch on the change of the instances in each version, for a wider range of system sizes with 32, 64 and 128 instances. The rolling upgrade and its control are coded in Java in AWS, and the algorithm, the bounding, and the optimisation are coded in R on a desktop computer. In both settings, the system options and the associated failure probabilities come from our previous tests and those reported in [3].

In Fig. 2 we compare the expected completion time from the model, the upper bound, and the average completion time from experiments including the loss of instances. It is visible that our model can capture the change of completion time. The bound can also reflect the change clearly, but as we can see in A and C when G becomes big enough the lines are almost horizontal for the bound. We call this phenomenon *saturation*, but the saturation does not apply for the probabilities as shown in B and D. The bound enters the saturation always earliest. This is the reason why the bound can be used for optimisation, since in most cases the bound can reflect the change of real completion time very well. Even if a sub-optimal solution is selected via the bound for big G, the errors must be very small. We test different numbers of options to observe the differences of Pareto sets and the computation times taken by $\mathcal{M}_1$ and $\mathcal{M}_2$ in E. The number of options is either 9 or 36 for the experiments with 32 or 64 instances, and the results are the average of many repeats. The cost values and the probabilities are randomly generated within the ranges. In order to eliminate the impact from different computers, we show the average coverage of $\mathbb{S}^*$ onto $\mathbb{S}$, and the ratio of the computation time of $\mathcal{M}_1$ to $\mathcal{M}_2$. Note that we only measure the time used for finding the Pareto sets rather than finding an exact optimum. It is easy to see that the coverage is high, but reducing as the search space size increases. While, the computation time taken by $\mathcal{M}_1$ is much shorter than $\mathcal{M}_2$, and the ratio becomes lower for large search space.

To show the difference between $\mathbb{S}^*$ and $\mathbb{S}$, the difference of optimum and sub-optimum, and the reason of obtaining a sub-optimum, one experiment is performed with fixed cost values, 24 options, and fixed probabilities for 32 instances. There are

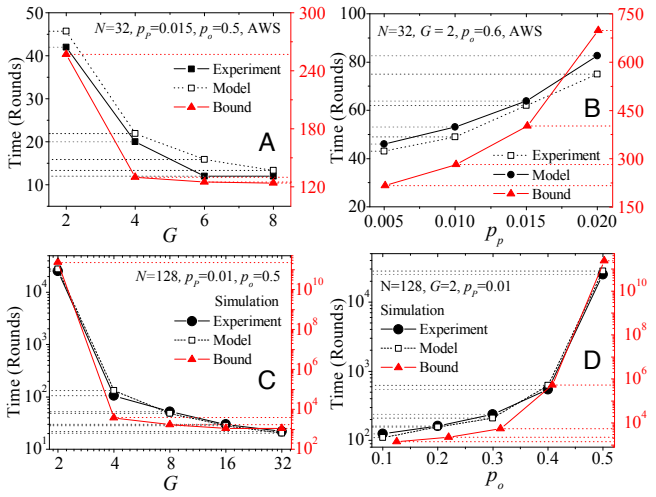

Fig. 2. Experiment results from AWS(A and B) and simulation (C and D). The left axis and the line marked by triangles in red are for the bound.

9 solutions missing from $\mathbb{S}^*$ compared to $\mathbb{S}$. We search for 24 optima under different weights in a brute-force way (the search space is not too huge to search in this way). There are only one different optimum. In $\mathbb{S}$ this optimum has $G = 18$ on $\alpha = 0.8$, $\beta = 0.1$, and $\gamma = 0.1$, and results in $T = 5.15$, $L = 18.3$, and $C = 35$, while in $\mathbb{S}^*$ the corresponding one is with a smaller granularity $G = 16$, and results in $T = 5.46$, $L = 16.3$, and $C = 35$. In terms of the objective function f, the one in $\mathbb{S}^*$ is a sub-optimum, but the expected completion time is only 0.29 round longer than the corresponding one in $\mathbb{S}$. Thus, the experiments testify that most optima can be found by $\mathcal{M}_1$, 1 lost out of 24, while the only sub-optimum is very close to the optimum. We should also notice that such a big G implies the loss of more than a half availability, and hence an operator will rarely set such a big α. In F of Fig. 3 we plot all completion time gaps in the saturation for all 24 options. The sub-optimum above is in the circle. The three blue gaps are the biggest but less than 10 rounds, and are possibly replaced by the neighbours with other options, because a solution may be replaced by another one with smaller value of objective function for specific weights.

VIII. Conclusion

In this paper, we discuss the optimisation of rolling upgrade on cloud platforms, when failures can occur. To achieve our goal, we have provided: 1) The formulations of a multi-objective optimisation, named the TCL problem; 2) Stochastic modelling of rolling upgrade; 3) A technique of quickly solving the TCL problem via the upper bound. The experiment results demonstrate that the prediction using the model is fairly accurate, and that the proposed optimisation method can find most optima much faster and the error is small if there is an sub-optimum.

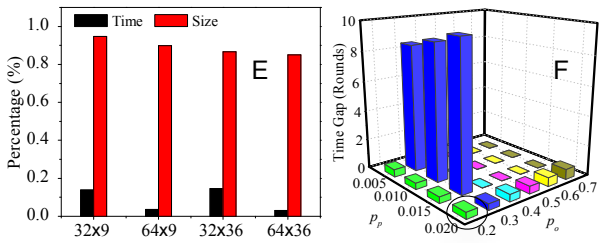

Fig. 3. In E, Time is the ratio of the time used for finding a Pareto set by $\mathcal{M}_1$ to that for $\mathcal{M}_2$. Size is the ratio of the size of Pareto set found by $\mathcal{M}_1$ to that by $\mathcal{M}_2$. F is the gaps.

References

[1] E. A. Brewer, "Towards robust distributed systems (abstract)," in *PODC*. ACM, 2000.

[2] O. Crameri, N. Knezevic, D. Kostic, R. Bianchini, and W. Zwaenepoel, "Staged deployment in mirage, an integrated software upgrade testing and distribution system," in *SOSP*. ACM, 2007, pp. 221–236.

[3] T. Dumitraş and P. Narasimhan, "Why do upgrades fail and what can we do about it?: Toward dependable, online upgrades in enterprise system," in *Middleware*, 2009, pp. 18:1–18:20.

[4] [Online]. Available: http://techblog.netflix.com/2012/06/asgard-web-based-cloud-management-and.html

[5] T. Dumitras, P. Narasimhan, and E. Tilevich, "To upgrade or not to upgrade: Impact of online upgrades across multiple administrative domains," in *OOPSLA*. ACM, 2010, pp. 865–876.

[6] S. Ajmani, B. Liskov, and L. Shrira, "Modular software upgrades for distributed systems," in *ECOOP*, Jul. 2006.

[7] S. Ajmani, "Automatic software upgrades for distributed systems," Ph.D. thesis, MIT, Sep. 2004.

[8] B. Zimmer, C. Dropmann, and J. U. Hanger, "A systematic approach for software interference analysis," in *ISSRE*. IEEE, 2014, pp. 78–87.

[9] A. Iosup, "Iaas cloud benchmarking: approaches, challenges, and experience," in *HotTopics*, 2013, pp. 1–2, invited lecture.

[10] V. Debroy and W. E. Wong, "Insights on fault interference for programs with multiple bugs," in *ISSRE*. IEEE, 2009, pp. 165–174.

[11] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for qos-aware clouds," in *EuroSys*. ACM, 2010, pp. 237–250.

[12] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. John Wiley and Sons, 2001.

[13] R. Pietrantuono, S. Russo, and K. S. Trivedi, "Software reliability and testing time allocation: An architecture-based approach," *IEEE Transaction on Software Engineering*, vol. 36, no. 3, pp. 323–337, 2010.

[14] W. Sun, Y. Zhang, C. Yu, X. Défago, and Y. Inoguchi, "Hybrid overloading and stochastic analysis for redundant real-time multiprocessor systems," in *SRDS*, 2007, pp. 265–274.

[15] D. Sun, L. Bass, A. Fekete, V. Gramoli, A. Tran, S. Xu, and L. Zhu, "Quantifying failure risk of version switch for rolling upgrade on clouds," in *Proceedings of International Conference on Big Data and Cloud Computing*, 2014.

[16] V. Gramoli, L. Bass, A. Fekete, and D. Sun, "Rollup: Non-disruptive rolling upgrade," The University of Sydney, Tech. Rep. 169, 2015.

[17] D. Sun, "An upper bound of absorbing markov models for modelling and analysis of software system reliability," SSRG, NICTA, Tech. Rep. 1833-9646-8559, 2015.

[18] [Online]. Available: https://aws.amazon.com/

[19] [Online]. Available: https://wiki.openstack.org/wiki/Heat

[20] P. Gill, N. Jain, and N. Nagappen, "Understanding network failures in data centres: Measurement, analysis, and implications," in *SIGCOMM*, 2011.

[21] NIST, "Common vulnerability scoring system (cvss)," 2014. [Online]. Available: http://nvd.nist.gov/cvss.cfm

[22] X. Xu, L. Zhu, L. Bass, I. Weber, and D. Sun, "Pod-diagnosis: Error diagnosis of sporadic operations on cloud applications," in *DSN*, 2014.

[23] J. Branke, K. Deb, K. Miettinen, and R. Slowinski, *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer, 2008.