Improving Cost-Efficiency through Failure-Aware Server Management and Scheduling in Cloud*

Laiping Zhao[1] and Kouichi Sakurai[2]

[1] School of Computer Software, Tianjin University, China
[2] Department of Informatics, Kyushu University, Japan
zhaolaiping@gmail.com, sakurai@csce.kyushu-u.ac.jp

Abstract. We examine the problem of managing a server farm in a cost-efficient way that reduces the cost caused by server failures, according to an Infrastructure-as-a-Service model in cloud. Specifically, failures in cloud systems are so frequent that severely affect the normal operation of job requests and incurring high penalty cost. It is possible to increase the net revenue through reducing the energy cost and penalty by leveraging failure predictiors. First, we incorporate the malfunction and recovery states into the server management process, and improve the cost-efficiency of each server using failure predictor-based proactive recovery. Second, we present a revenue-driven cloud scheduling algorithm, which further increases net revenue in collaboration with server management algorithm. The formal and experimental analysis manifests our expected net revenue improvement.

Keywords: Net Revenue, Server Management, Scheduling, Failure Prediction.

1 Introduction

With the infrastructure-as-a-service (IaaS) model in cloud computing, a business is enabled to run jobs on virtual machine (VM) instances rented from the infrastructure service providers in a pay-as-you-go manner. As shown in Figure 1, multiple applications are consolidated to share the same physical server through virtualization technologies. VM instances are offered from a diversified catalog with various configurations. Jobs are encapsulated into VMs, and customers can start new VMs or stop unused ones to meet the increasing or decreasing workload, respectively, and pay for what they use thereafter. In this process, customers do not have full control over the physical infrastructure. Instead, the provider sets a resource management policy determining the physical servers for starting VMs. VM instances are commonly provided under a Service Level Agreement (SLA), which gurantees the service quality, and a penalty is punished on the provider if SLA is violated. For example, Amazon EC2 claims that the customer is eligible to receive a service credit equal to 10% of their bill, if the annual uptime percentage is less than 99.95% during a service year. During the job execution, a VM may migrate from one server to another according to the policy.

* This work is based on "On Revenue Driven Server Management in Cloud", by L. Zhao and K. Sakurai, which appeared in Proc. of 2nd International Conference on Cloud Computing and Service Science, Portugal, April 2012.

I. Ivanov et al. (Eds.): CLOSER 2012, CCIS 367, pp. 23–38, 2013.

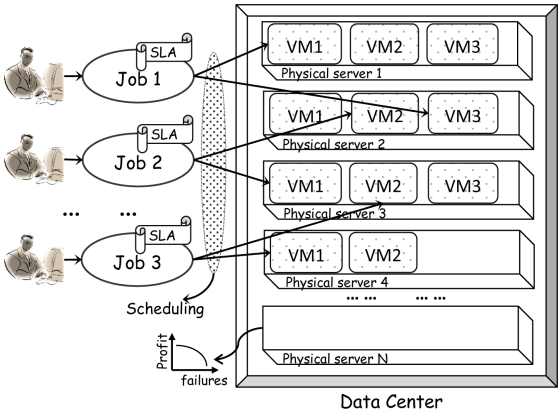

Fig. 1. The cloud IaaS model

SLA violation, that is failing to meet the availability level, is generally caused by inadequate resources or server failures. Managing SLA violations caused by inadequate resources has been studied in [1]. However, few of them consider reducing the SLA violation cost caused by server failures. As the system scale continues to increase, problems caused by failures are becoming more severe than before ([2], or [3, chap.7]). For example, according to the failure data from Los Alamos National Laboratory (LANL), more than 1,000 failures occur annually at their system No.7, which consists of 2014 nodes in total [2], and Google reports 5 average worker deaths per MapReduce job in March 2006 [4].

The frequent failures as well as the resulting SLA violation costs lead to a practical question: *how to improve the cost efficiency of service providing?* In this paper, we aim to explore a new cost-efficient way to manage the cloud servers by leveraging the existed failure prediction methods. The basic idea is that, a failure-prone server should reject a new arrived job, or move a running job to another healthy server, then proactively accept manual repairs or rejuvenate itself to a healthy state. Our contributions mainly fall into three parts:

- We analyze the cost for job execution and SLA violation, and propose a novel server management model by combining the failure prediction together with proactive recovery into server state transitions.
- We design an adaptive net revenue-based decision making policy that dynamically decides whether accepting a new job request or not, and whether moving the running job to another healthy server or not while achieving high cost efficiency.
- We further increase the cost-efficiency through a collaboration of server management algorithm and scheduling algorithm, i.e., *MaxReliability*. The experimental results manifest the revenue improvement.

2 Related Work

Feasibility of our approach depends on the ability to anticipate the occurrence of failures. Fortunately, a large number of published works have considered thecharacteristics

of failures and their impact on performance across a wide variety of computer systems. These works include either the fitting of failure data to specific distributions or have demonstrated that the failures tend to occur in bursts [5],[6]. Availability data from BONIC is modeled with probability distributions [7], and their availabilities are restricted by not only site failures but also the usage patterns of users. Fu and Xu [8] propose a failure prediction framework to explore correlations among failures and forecast the time-between-failure of future instances. Their experimental results on LANL traces show that the system achieves more than 76% accuracy. In addition to processor failures, failure prediction is also studied on hard disk drives [9]. As a survey, Salfner et al. [10] present a comprehensive study on the online failure prediction approaches, which can be split into four major branches of the type of input data used, namely, data from failure tracking, symptom monitoring, detected error reporting, and undetected error auditing. In each branch, various methodologies are applied, for instance, bayesian network, machine learning, time series analysis, fuzzy logic, markov chain, and pattern recognition.

Economic cost for constructing a data center has been studied in [11], [12], [3], which provide us with a deep understanding of cloud system cost. The revenue maximization problems discussed in the literature[13][14][15], are quite close to our work. Mazzucco et al. [15] measure the energy consumed by active servers, and maximize the revenue by optimizing the number of servers to run. Macías et al. [14] present an economically enhanced resource manager for resource provisioning based on economic models, which supports the revenue maximization across multiple service level agreements. Maximizing of the quality of users' experiences while minimizing the cost for the provider is studied in [16]. And Fitó et al. [13] model the economic cost and SLA for moving a web application to cloud computing. The proposed *Cloud Hosting Provider* (CHP) could make use of the outsourcing technique for providing scalability and high availability capabilities, and maximize the providers' revenue. In contrast to their proposals, our goal is to improve the cost efficiency of servers by leveraging the failure prediction methods.

3 Policy for Server Management

3.1 Cloud Server Management

A physical server is described with five states as follows:

IDLE: There are no VMs executing on the server.

RUNNING: The server is executing some VM(s).

TERMINATED: The server successfully finishes jobs, then recycles the memory and clears the disk.

MALFUNCTION: A failure occurs, and the server breaks down.

RECOVERY: Troubleshooting, which could be a simple reboot or repair by a skilled staff.

Figure 2 illustrates the above states and their state transitions for a physical server. We incorporate both the MALFUNCTION and RECOVERY into the states due to the common failures. Although failures may occur at anytime, the probability of failure

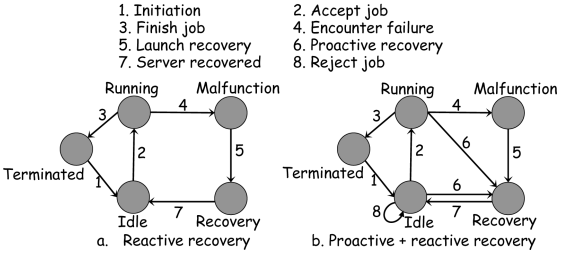

Fig. 2. The state transitions of (a) Reactive recovery and (b) Proactive-Reactive recovery

occurrence in the TERMINATED or IDLE state is far less than in the RUNNING state. Therefore, a single in-degree to the MALFUNCTION state is exploited.

Initially, when a new physical server joins a server farm, or an existing server has finished all deployed VMs and refreshed his status, the server enters the IDLE state and becomes ready for serving a next job.

Reactive Recovery. When a job arrives, the server starts a required VM and accepts the job without hesitation, then comes into the RUNNING state. In case of a successful execution, the job is completed, and the server enters the TERMINATED state. After clearing the memory and disk, the server returns to the IDLE state. If a failure occurs, the server comes into the MALFUNCTION state. Certain recovery methods, e.g., repair, rebooting, would be activated to fix the malfunction, then the server returns to the IDLE state. Note that the recovery could be launched by a skilled staff or automatically activated by a tool like watchdog.

Proactive-Reactive Recovery. Proactive recovery is a useful tool for improving the system efficiency and reducing the economic cost. However, the effects of proactive recovery heavily depend on the failure prediction accuracy, which is still in the rough primary stage currently. Therefore, we employ a hybrid approach based on both proactive recovery and reactive recovery here. An architectural blueprint for managing server system dependability in a proactive fashion is introduced in Polze, Troger, and Salfner (2011).

When a job arrives, the server can: 1. accept the job and change to RUNNING (step 2 in Figure 2(b)); 2. reject the job and stay in IDLE (step 8); 3. reject the job and activate a proactive recovery operation if a failure is predicted (step 6). To assure a positive net revenue, we devise a utility function to handle such decision making problems.

In the first case, if the server comes into the RUNNING state, there are three further possible transitions coming out from the RUNNING state: 1a. if a failure is anticipated during RUNNING, move all running VMs and proactively launch the recovery (step 6); 1b. if a failure occurs without warning, the server reactively comes into the MALFUNCTION state (step 4); 1c. if there are no failures, complete the job successfully (step 3). A similar utility based function is also employed here for the proactive recovery operation. In the second and third cases, the server needs to decide whether to stay in IDLE state, or activate a proactive recovery after a job rejection. This is reasonable

Table 1. List of notations

Notations	Definition
Prc_i	Price of the VM instance i ($/hour).
Prc_{egy}	Price of energy consumption. ($/kw.h)
U_{SIO_i}	Fixed cost of a VM i.
P_i	Energy consumption per time unit for VM i.
$Ucoe_i$	Task execution cost per time unit. $Ucoe_i = U_{SIO_i} + Prc_{egy}P_i$
T_{vm}	Job execution time, or contract life.
Coe_i	Total task execution cost of VM i. $Coe_i = Ucoe_i T_{vm}$
Pen	Penalty for SLA violations.
T_M	Time spent on MALFUNCTION state.
T_R	Time spent on RECOVERY state.
P_{SLA}	The percentage of total bill that the provider has to refund.
P_{fail}	Probability of failures.
$Cmig$	VM migration cost.
T_{vm}^{rmn}	VM's remaining lifetime.
P_{fail}^{mig}	Probability of failures for a migrated VM.

because a negative net revenue may be expected from a long-running job, whereas a positive net revenue is expected from a short-running job. If the rejected job is a normal or small size one, then the server activates a proactive recovery, otherwise stays in the IDLE state.

3.2 Net Revenue

Our net revenue model is similar to those used in the literature [14] [13] except that we do not consider the situation of outsourcing a application to a third-party and hence there is no outsourcing cost [13]. Notations are listed in Table 1.

Price (Prc): is the amount of money that a customer has to pay if a cloud provider finishes his job successfully. It usually takes the time piece as the unit. For instance, Amazon EC2 standard small instance charges 0.085$ per hour. *Cost of execution* (Coe): is the amount of money that a cloud provider has to spend on maintaining a healthy VM as well as a physical server for providing service. As the service providing is the major source of profit, any cost for maintaining such service providing will be considered as the part of the total costs, which typically includes fixed costs (e.g., site infrastructure capital costs, IT capital costs) and variable costs (e.g., energy costs, operating expenses). *Penalty* (Pen): is the amount of money that a provider has to pay if the SLA is violated. Denote by T_{vm} the working time (i.e., contract life or job execution time), the net revenue obtained from deploying VM i is computed as below,

$$Rvu_i = Prc_i \cdot T_{vm} - Coe_i - Pen_i \tag{1}$$

The prices of different VM instances have been clearly announced by the providers, and can be publicly accessed. For the cost of execution, we have $Coe_i = Ucoe_i \cdot T_{vm} = (U_{SIO_i} + Prc_{egy} \cdot P_i) \cdot T_{vm}$, where U_{SIO_i} comprises site infrastructure capital costs (Sic), IT capital costs (Icc) and operating expenses (Ope). Prc_{egy} denotes the hourly energy cost (Enc), and P_i denotes the consumed energy [3][11][12]. For the penalty,

	Run	Idle	Termi.	Malfunc.	Reco.
Running	0/0	0/0	P'_{02}/P_{02}	P'_{03}/P_{03}	$0/P_{04}$
Idle	$1/P_{10}$	$0/P_{1}$	0/0	0/0	$0/P_{14}$
Terminated	0/0	1/1	0/0	0/0	0/0
Malfunction	0/0	0/0	0/0	0/0	1/1
Recovery	0/0	1/1	0/0	0/0	0/0

Fig. 3. The state transition probability for Reactive/Proactive-reactive recovery

Reactive recovery		Revenue
Path 1	IDLE - RUNNING - TERMINATE - IDLE	A_R
Path 2	IDLE - RUNNING - MALFUNCTION - RECOVERY - IDLE	$-B_R$
Proactive-Reactive recovery		
Path 1	IDLE - RUNNING - TERMINATE - IDLE	A_{PR}
Path 2	IDLE - RUNNING - MALFUNCTION - RECOVERY - IDLE	$-B_{PR}$
Path 3	IDLE - RUNNING - RECOVERY - IDLE	C_{PR}
Path 4	IDLE - IDLE	0
Path 5	IDLE - RECOVERY - IDLE	$-D_{PR}$

Fig. 4. The server running paths and their corresponding net revenues

we have $Pen_i = Prc_i \cdot T_{vm} \cdot P_{SLA}$, where P_{SLA} denotes the fraction of total bill that the provider has to refund.

3.3 Expected Net Revenue

Next we compute the expected net revenue from providing service or possible losses from server failures. Figure 3 shows the probabilities of state transitions for both the reactive recovery and proactive-reactive (Figure 2). Figure 4 shows all the state transition paths and the corresponding revenues for both of them.

In reactive recovery, the cloud server could obtain positive net revenue from job execution in path 1 (denoted as $A_R = (Prc - Ucoe) \cdot T_{vm}$). While in path 2, the cloud server not only obtains nothing due to the failure, but also has to pay the penalty and losses the execution cost. Let T_M and T_R be the time spent on MALFUNCTION and RECOVERY state respectively, then $B_R = U_{SIO}(T_{vm} + T_M + T_R) + Prc_{egy} \cdot P \cdot T_{vm} + Prc \cdot T_{vm} \cdot P_{SLA}$.

In the proactive-reactive recovery, the cloud server shows a similar situation in path 1 and path 2, that is $A_{PR} = A_R$ and $B_{PR} = B_R$, but with different probabilities. In path 3, a proactive recovery is activated during the running process. The running VMs are interrupted and moved to other healthy servers, where they subsequently proceed until finish. During this process, the cloud provider eventually gets the revenue from these jobs. The revenue generated in terms of the old server is computed based on the finished fraction of the total workload, denoted by P_{fnd}, therefore, we have $C_{PR} = P_{fnd}(Prc - Ucoe)T_{vm} - U_{SIO}T_R$. In path 4 and 5, a job rejection operation only implies a local server's decision, and the rejected job is eventually accepted by another healthy server from the perspective of cloud provider. Thus, there are no losses caused from the job rejection. And the recovery cost spent on the proactive recovery operation in path 5 is: $D_{PR} = U_{SIO}T_R$.

According to Figure 3 and Figure 4, the expected net revenue generated by reactive recovery is:

$$Rvu_R = A_R P'_{02} - B_R P'_{03} \tag{2}$$

The expected net revenue generated by proactive-reactive recovery is:

$$Rvu_{PR} = A_{PR}P_{10}P_{02} - B_{PR}P_{10}P_{03} + C_{PR}P_{10}P_{04} - D_{PR}P_{14} \tag{3}$$

Theorem 1. $Rvu_{PR} > Rvu_R$.

Proof.

$$\begin{aligned} &Rvu_{PR} - Rvu_R = PrcT_{vm} \times \\ &\underline{[P_{SLA}(P_{03}^{'} - P_{10}P_{03}) + (P_{10}P_{02} - P_{02}^{'}) + P_{fnd}P_{10}P_{04}]} + \\ &U_{SIO}[T_{vm}(\underline{P_{03}^{'} - P_{10}P_{03} - P_{fnd}P_{10}P_{04} + P_{02}^{'} - P_{10}P_{02}}) + \\ &T_M(\underline{P_{03}^{'} - P_{10}P_{03}}) + T_R(\underline{P_{03}^{'} - P_{10}(P_{03} + P_{04}) - P_{14}})] + \\ &Prc_{egy}PT_{vm}[\underline{P_{02}^{'} - P_{10}P_{02} + P_{03}^{'} - P_{10}P_{03} - P_{fnd}P_{10}P_{04}}] \end{aligned} \tag{4}$$

With the two prerequisites, $P_{10}P_{02} = P'_{02}$ and $P_{10} \cdot (P_{03} + P_{04}) + P_{11} + P_{14} = P'_{03}$, we have,

- $P_{SLA}(P'_{03} - P_{10}P_{03}) + (P_{10}P_{02} - P'_{02}) + P_{fnd}P_{10}P_{04} > 0$
- $T_{vm}(P_{03}^{'} - P_{10}P_{03} - P_{fnd}P_{10}P_{04} + P_{02}^{'} - P_{10}P_{02}) + T_M(P_{03}^{'} - P_{10}P_{03}) + T_R(P_{03}^{'} - P_{10}(P_{03} + P_{04}) - P_{14}) > 0$
- $P_{02}^{'} - P_{10}P_{02} + P_{03}^{'} - P_{10}P_{03} - P_{fnd}P_{10}P_{04} > 0$

Hence, the theorem is established. □

3.4 Decision Making

The server state transitions contain three decision making points. The first one is to decide whether to accept or reject a new arriving job, and followed by a further decision is on whether or not to activate proactive recovery if the job is rejected. The third one is to decide whether to activate a proactive recovery or continue the job execution when the job is under the RUNNING state. In our proposal, these decisions are made on behalf of physical servers based on the expected net revenue.

Accept or Reject a Job. A job arriving at a cloud server could be a new or rejected or failed or migrated job. After its lifetime (i.e., T_{vm}) is determined by user's specification or estimates, we can predict the probability of failures (i.e., P_{fail}) in this interval using associated stressors [17]. The possible net revenue obtained from accepting a job by server i can be computed as,

$$\overline{Rvu_i} = A_{PR}^i \cdot (1 - P_{fail}) \tag{5}$$

Accounting of malfunction losses during the middle of job execution consists of direct economic loss and indirect economic loss. A cloud provider would directly get a penalty from the SLA agreement, and he also has to afford the cost for recovery operation. The possible losses can be computed as,

$$\overline{Los_i} = B_{PR}^i \cdot P_{fail} \tag{6}$$

If $\overline{Rvu_i} > \overline{Los_i}$, the VM i will be deployed for execution, and if not, the VM i will be rejected. In other words, the VM i is accepted when the following condition is held,

$$P_{fail} < \frac{A_{PR}^i}{A_{PR}^i + B_{PR}^i} \tag{7}$$

In case of a migrated VM, additional cost is spent on VM migration. Let $Cmig$ be the cost for moving the VM from an old physical server to a new one, and T_{vm}^{rmn} be the remaining lifetime. We have,$\overline{Rvu_i^{mig}} = (A_{PR}^{i,rmn} - Cmig) \cdot (1 - P_{fail}^{mig}) = ((Prc_i - Ucoe_i) \cdot T_{vm}^{rmn} - Cmig) \cdot (1 - P_{fail}^{mig})$ and $\overline{Los_i^{mig}} = (B_{PR}^{i,rmn} + Cmig) \cdot P_{fail}^{mig} = (Pen_i + U_{SIO}(T_{vm}^{rmn} + T_M + T_R) + Prc_{egy} \cdot P \cdot T_{vm}^{rmn} + Cmig) P_{fail}^{mig}$

Let $\overline{Rvu_i^{mig}} > \overline{Los_i^{mig}}$, then the condition 7 is changed into,

$$P_{fail}^{mig} < \frac{A_{PR}^{i,rmn} - Cmig}{A_{PR}^{i,rmn} + B_{PR}^{i,rmn}} \tag{8}$$

Proactive Recovery or Not. Once a job is rejected, a cloud server further has two options of launching the proactive recovery or doing nothing. Denote by $\overline{T_{vm}}$ the average lifetime of all history VMs successfully completed on a server, and $P_{fail}^{\overline{T_{vm}}}$ the predicted probability of failures within next $\overline{T_{vm}}$ time. Then if $P_{fail}^{\overline{T_{vm}}} < P_{fail}$ (The right side of Inequality 7), the cloud server does nothing but waits for the next job. Otherwise, the server activates the proactive recovery. This is because failure probability increases over time. A next normal size job still can be accepted if $P_{fail}^{\overline{T_{vm}}} < P_{fail}$ is held. Note that it is possible that the server stays in a starvation state for a long time, if no short-running VM is dispatched to the server. In such case, activate the proactive recovery if the leisure time exceeds a pre-defined threshold.

Activate VM Migaration or Not. For a long-running VM, it is difficult to have an accurate prediction on the failure probability during that long time. Moreover, certain types of failures always come with some pathognomonic harbingers. It is a difficult to predict such failures without particular harbingers. Therefore, we also activate the failure prediction during the running process. And proactive recovery is launched when inequality 7 (inequality 8 if it is a migrated job) is violated for all the local VMs.

The VM migrates from a server i to a server j only because j can yield a greater net revenue. The expected revenue obtained from no migration is the same as $\overline{Rvu_i}$ (Equation 5), except that the failure probability (P_{fail}^{rmn}) is for the remaining lifetime (i.e., T_{vm}^{rmn}): $\overline{Rvu_i^{rmn}} = A_{PR}^{i,rmn} \cdot (1 - P_{fail}^{rmn}) = (Prc_i - Ucoe_i) \cdot T_{vm}^{rmn} \cdot (1 - P_{fail}^{rmn})$. The expect net revenue obtained from a migration has been described as $\overline{Rvu_j{}^{mig}}$. Let $\overline{Rvu_j^{mig}} > \overline{Rvu_i^{rmn}}$, we have,

$$P_{fail}^{mig} < \frac{A_{PR}^{i,rmn} \cdot P_{fail}^{rmn} - Cmig}{A_{PR}^{i,rmn} - Cmig} \tag{9}$$

Therefore, a new server j whose failure probability (i.e., P_{fail}^{Mig}) follows both Inequality 8 and 9 will be selected to execute the migrated VM. In case of more than one server meets this condition, the VM migrates to the one with maximum reliability to proceed. If no appropriate processors are found, maintain the VM at the original server until finish or failure.

Algorithm 1. Algorithm for server management.

```
1  [Parameters]
2  double f_p {the predicted failure probability}
3  double P_fail {the probability that ensures the job acceptance}
4  double P_fail^Mig {the probability that ensures condition 9}
5  double P_fail^rmn {the failure probability in the remaining time}
6  integer T̄ {the average job execution time}
7  STATE state = IDLE {server state, initialized with IDLE}
8  upon receive(job)
9  if state = IDLE then  job_submit(job);
10 upon receive(job, P_fail^Mig)                          /* received a migrated job */
11 f_p = fail_predict(T_job^rmn); P_fail = fail_expect(T_job)   /* ensure condition 8 */
12 if f_p < P_fail^Mig and f_p < P_fail and state = IDLE then return true;
13 else return false;
14 procedure job_submit(job)
15 T_job = time_estimate(); f_p = fail_predict(T_job); P_fail = fail_expect(T_job);
16 if f_p < P_fail then                                   /* ensure positive revenue */
17    job_execute(job); state = RUNNING;
18 else if T_job ≤ T̄ then                                 /* proactive recovery */
19    proactive_recovery(job); state = RECOVERY ;
20 else                                                   /* wait next job */
21    state = IDLE ;
22 procedure job_execute(job)
23 if failed = true then                                  /* reactive recovery */
24    state = MALFUNCTION; return;
25 P_fail^rmn = fail_predict(T_rmn);
26 if P_fail^rmn ≥ P_fail then                            /* negative revenue */
27    compute P_fail^Mig using 9; reschedule(job, P_fail^Mig);
28    wait until find another processor
29       migrate(job);  proactive_recovery(job);  state = RECOVERY;
30 procedure fail_expect(T_job)
31    return P_fail using 7;
```

3.5 Algorithm Description

Algorithm 1 presents the detail description for server management. The migrated job and other job requests are handled by *receive(job, P)* and *receive(job)* respectively. If it is a migrated job, the server will reply with an affirmative answer if its predicted failure probability follows both inequality 8 and 9, otherwise with a negative answer (line 10-13).

We use the function *job_submit()* to decide whether to accept or reject the job. If the job is rejected and its execution time is less than the average level of job execution time, activate a recovery operation directly (line 18-19). Otherwise, do nothing but wait for the next job request (line 21). Function *job_execute()* activates the proactive recovery after all VMs migrated to other servers because of a sudden higher failure probability ($P_{fail}^{rmn} \geq P_{fail}$).

4 Revenue-Driven Scheduling

Since proactive recovery contributes to avoid the possible penalties caused by server failures, the server management algorithm is able to increase the revenue for cloud

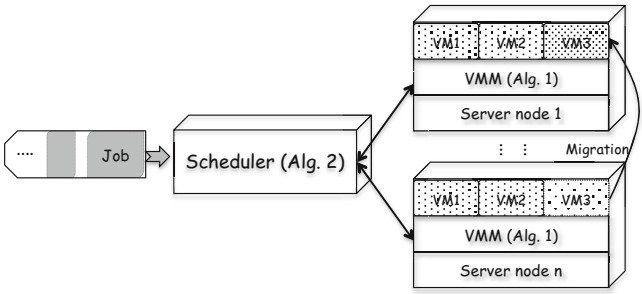

Fig. 5. The scheduling framework

provider. In fact, the revenue could be further increased through collaborative use of server management and cloud scheduling algorithm. Figure 5 shows the framework. Algorithm 1 is implemented at the Virtual Machine Monitor (VMM), which takes in charge of failure estimation, VM migration and proactive recovery. Below, we discuss how to schedule the VMs for increasing cloud provider's revenue.

4.1 MaxReliability

As physical servers perform high heterogeneity in failure probability, the probability of state transition from *RUNNING* to *MALFUNCTION* would be different for different servers (i.e., P_{03} in Figure 3).

Theorem 2. *Suppose server i is more reliable than server j. For performing the same VM, the expected net revenue yielded by i is greater than j: $Rvu^i_{PR} > Rvu^j_{PR}$.*

Proof. Because server i is more reliable than j, it means the probability of state transition from *IDLE* to *RUNNING*, then to *TERMINATED* for server i is greater than j. Thus, according to Figure 3, we have $P^i_{10} > P^j_{10}$, $P^i_{10}P^i_{02} > P^j_{10}P^j_{02}$, $P^i_{14} < P^j_{14}$, and $P^i_{03} + P^i_{04} < P^j_{03} + P^j_{04}$. Note that, for the predicted failure probability: $f^i_p = P^i_{03} + P^i_{04}$ and $f^j_p = P^j_{03} + P^j_{04}$.

Suppose the failure detection accurancy is: $\alpha = No.of detected failures / No.of failures$, then $P^i_{03} = (1-\alpha)f_p$ and $P^i_{04} = \alpha f_p$. As $P^i_{02} + P^i_{03} + P^i_{04} = 1$, and regrading Equation 3, we have,

$$\begin{aligned} Rvu^i_{PR} &= A_{PR}P^i_{10}P^i_{02} - B_{PR}P^i_{10}P^i_{03} + C_{PR}P^i_{10}P^i_{04} - D_{PR}P^i_{14} \\ &= P^i_{10}[A_{PR}(1-f^i_p) - f^i_p(B_{PR}(1-\alpha) - C_{PR}\alpha)] - D_{PR}P^i_{14} \\ &= P^i_{10}[A_{PR} - f^i_p(A_{PR} + B_{PR}(1-\alpha) - C_{PR}\alpha)] - D_{PR}P^i_{14} \end{aligned}$$

Because $f^i_p < f^j_p$, $A_{PR} + B_{PR}(1-\alpha) - C_{PR}\alpha > 0$, then $A_{PR} - f^i_p(A_{PR} + B_{PR}(1-\alpha) - C_{PR}\alpha) > A_{PR} - f^j_p(A_{PR} + B_{PR}(1-\alpha) - C_{PR}\alpha)$. Likewise, because $P^i_{14} < P^j_{14}$, then $D_{PR}P^i_{14} < D_{PR}P^j_{14}$. Hence, we have $Rvu^i_{PR} > Rvu^j_{PR}$. □

Theorem 2 suggests that scheduling VMs on reliable servers could increase cloud provider's revenue. Therefore, a natural way of cloud scheduling is always dispatching

an incoming VM to the most reliable server. We call the rule as *MaxReliability*. Algorithm 2 shows the details: Upon receiving a job by the scheduler, either a new job or a migrated job, iterate over all servers and predict the probability of failure. Then dispatch the job to the server with maximum reliability (Line 1-5). Whenever a VM is completed, because capacity is freed on the host server (e.g., s_k), VMs located on less reliable servers could migrate to s_k for increasing revenue. Iterating over all busy servers (Line 7), we find the job j with least reliability, and j's capacity requirement is able to be satisfied by s_k available capacity. Then, move j to s_k (Line 13). The process is repeated until s_k is full. Note that the migration comes with overhead (i.e., C_{mig}), hence a VM is migrated only when the difference on failure probability, i.e., $f_p^{s,j_{min}} - f_p^{s_k,j_{min}}$, is greater than a threshold ϵ (Line 13-14).

Algorithm 2. Algorithm for cloud scheduler: *MaxReliability*.

input : S {*the servers set*}
output : schedule for jobs

1 **upon** *receive(job)* /* Either a new or a migrated job */
2 **for** $s \in S$ **do**
3 f_p^s = *fail_predict*(T_{job});
4 $s_{max} = max\{(1 - f_p^s)\}$;
5 *send(job,*s_{max}*)*; /* Send the job to the server with max reliability */
6 **upon** *free*(s_k) /* Part capacity of s_k is freed due to completion of jobs */
7 *flag = 1.0* ;
8 **for** $s \in S$ && $s.state == RUNNING$, *except* s_k **do**
9 **for** *job* j *running on* s **do**
10 f_p^{s,rmn_j} = *fail_predict*(T_j^{rmn});
11 **if** $flag > f_p^{s,rmn_j}$ && $j.capacity < s_k.availablecapacity$ **then**
12 $flag = f_p^{s,rmn_j}$; $j_{min} = j$; /* Find the most fragile job j_{min} */
13 **if** $f_p^{s,j_{min}} - f_p^{s_k,j_{min}} > \epsilon$ **then**
14 *move* j_{min} *to* s_k; /* Move the job with least reliability to s_k */

4.2 Combined with Energy-Saving

Let us revisit the equation for computing revenue: $Rvu_i = Prc_i \cdot T_{vm} - Coe_i - Pen_i$. To increase the revenue, there are several possible ways regarding the equation. For example, cloud provider could increase Rvu_i through providing flexible pricing functions (i.e., Prc_i), reducing execution cost Coe_i or reducing Pen_i. Flexible pricing function has been discussed in the literature [18], [19], [20], and also employed in practice (e.g., the spot instance at Amazon). In fact, pricing functions is based on long-term service providing. For a single VM, after it is submitted, the price is fixed no matter which pricing functions is applied. Therefore, we do not address the pricing function here, but

tend to increase the Rvu by reducing Coe_i and Pen_i.

According to equation $Coe_i = (U_{SIO_i} + Prc_{egy} \cdot P_i) \cdot T_{vm}$, reducing energy cost (i.e., P_i) is the most likely way to reduce Coe_i, because U_{SIO} belongs to fixed-asset investment. Thus, we explore to increase cloud provider's revenue through reducing energy cost and penalty simultaneously. Fortunately, reducing energy cost does not conflict with reducing penalty. Scheduling VMs to reliable servers could avoid penalty, and consolidating several VMs on a server could reducing the energy cost. A natural combination of them would be, consolidating VMs on the least number of reliabile servers. That is, a high reliable server could contribute to reduce the penalty, while consolidation is able to reduce energy cost. For example, Mastroianni et al. [21] present a decentralized solution for VM consolidation. That is, if the CPU utilization of a server is below certain threshold, VMs on other servers could be migrated to it. While if the CPU utilization of a server is above certain threshold, VMs on it can be migrated to other underloaded servers. The improved version with considering penalty cost could be: if the CPU utilization of a server is below certain threshold, VMs on the least reliable and overloaded server could be migrated to it. While if the CPU utilization of a server is above certain threshold, VMs on it can be migrated to the first available server with maximum reliabily.

5 Experiments

5.1 Simulation Environment

Server Farm. We simulate a server farm with 20 physical servers, which can provide seven different types of VM instances, corresponding to the seven types of instances from Rackspace Cloud Servers [22]. The processing speed of each server is supposed to be the same, and is initialized with eight cores, with each is of $2.4GHz$.

Job. We simulate a large number of jobs ranging from 1000 to 6000, for maximizing the utilization of all servers. Through this way, the cost for maintaining a server could be fairly shared among all the VMs deployed on it, and leading to a positive net revenue. This is also reasonable in practice because cloud providers commonly design policies to optimize the minimum number of active servers for reducing the energy cost, thereby resulting in a high utilization at each active server (Mastroianni, Meo, and Papuzzo, 2011; Mazzucco et al., 2010b).

Allowing two instructions in one cycle, the workload of each job is evenly generated from the range: $[1, 6] \times 204,800 \times 2^i$ million instructions, where $0 \leq i \leq 6$ and $i \in Z$, represents the type of VM instance this job requires.

Scheduling. Jobs are placed on cloud servers using a First-Come-First-Served (FCFS) algorithm. Scheduling priority is supported, and follows the sequence: *migrated job* $>$ *failed job* $>$ *rejected job* $>$ *unsubmitted job*. The rejected or failed jobs will not be scheduled on the same server at the second time, because it is probably rejected or failed again.

Failures. Failures are considered from two dimensions. The first dimension concerns the time when failures occur. In our experiments, failures are injected to servers following a Poisson distribution process with $\lambda = [1, 4]/\theta \times 10^{-6}$, where $\theta \in [0.1, 2]$. According to the Poisson distribution, the lengths of the inter-arrival times between failures follow the exponential distribution, which is inconsistent with the observations that the time between failures is well modeled by a Weibull or lognormal distribution [2]. The deviations arise because an attempt to repair a computer is not always successful and failures recur a relatively short time later [23]. Implementing a real failure predictor is out of the range of this paper, and we alternatively consider different failure prediction accuracy in evaluations.

The second dimension concerns the repair times. If an unexpected failure occurs, the server turns into a MALFUNCTION state immediately, and followed by recovery operations. As discussed in [2], the time spent on recovery follows a Lognormal distribution process, which is defined with a mean value equaling to 90 minutes, and $\sigma = 1$.

Price and Server Cost. The prices for all seven types of VM instances are set exactly the same as the ones from Rackspace Cloud Servers [22], that is $Prc = 0.015\$ \times 2^i$, where $0 \leq i \leq 6$ and $i \in Z$.

The capital cost is roughly set at 8000$ per physical server, which is estimated based on the market price of *System x 71451RU server* by IBM [24]. The price of electricity is set at 0.06$ per kilowatt hour. We suppose that the power consumption of an active server without any running VMs is 140 Watts. Additional power ranging from 10 Watts to 70 Watts is consumed by VMs corresponding to seven types of VM instances [16]. Suppose a server's lifetime is five years, and as the IT capital cost takes up 33% to the total management cost, we roughly spend 4300$ on a physical server per year with additional energy cost.

As modeling of migration costs is highly non-trivial due to second order effects migrations might have on the migrated service and other running services, we simplify migration costs as an additional 20% of the unexecuted workload without profit (i.e., 10% for the original server, and another 10% for the target server). A preliminary attempt on modeling migration cost is given in Breitgand, Kutiel, and Raz (2010).

Penalty. If a SLA is breached due to the provider's failing to complete jobs, the customer gets compensation by a rather high ratio of fine, which ranges from 10% to 500% of the user bill in the experiments (i.e., $P_{SLA} \in [0.1, 5]$). This is because SLA violations cause not only direct losses on revenue but also indirect losses, which might be much more significant (e.g., in terms of provider reputation).

5.2 Results

We choose the evaluation approach by comparing our proposed proactive-reactive model with the original reactive model. Experiments are conducted across a range of operating conditions: number of jobs, failure frequency (θ), P_{SLA}, and the accuracy of failure prediction. Accuracy implies the ability of the failure prediction methods, and is presented by both the false-negative (fn) and false positive (fp) ratio. Denote by

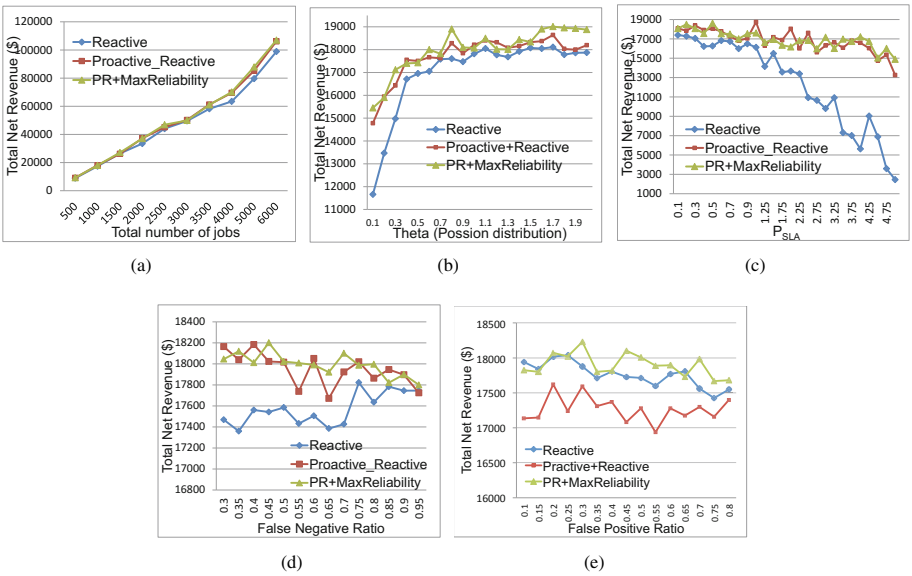

Fig. 6. The total net revenue: (a) under different No. of jobs; (b) under different qs (failure frequency); (c) under different SLA penalty percentages; (d) under different levels of false-negative ratio; (e) under different levels of false-positive ratio

$No(FN)$ the number of false-negative errors, $No(TP)$ the number of true-positive predictions, and $No(FP)$ the number of false-positive errors, then we have $fn = \frac{No(FN)}{No(FN)+No(TP)}$ and $fp = \frac{No(FP)}{No(FP)+No(TP)}$. Unless otherwise stated, the parameters are set with $jobnumber = 1000$, $\theta = 1$, $P_{SLA} = 0.1$, $fn = 0.25$ and $fp = 0.2$. Each experiment is repeated five times and the results represent the corresponding average values.

Figure 6 shows the total net revenue obtained by the reactive recovery, proactive-reactive recovery (PR) and PR+MaxReliability (PRM) from executing jobs under different operating conditions. Note that random selection is employed in both reactive recovery and proactive-reactive recovery (PR) for dispatching VMs to servers, while PRM uses *MaxReliability* for scheduling. Generally, net revenue yielded by the proactive-reactive model is greater than the reactive model, which is consistent with our analysis in Theorem 1. However, compared with random selection, *MaxReliability* does not yield much improvement on revenue. This is because, we have submitted so many VMs that nearly all servers are fully occupied. Thus the difference on revenue between them is not apparent. In particular, Figure 6(a) shows the net revenue as a function of the number of jobs. The difference on net revenue is increasing over the number of jobs, which is reasonable because more jobs come with more revenue.

Figure 6(b) shows the net revenue as a function of failure frequency. By the definition of λ, we know failure frequency decreases as θ increases. As shown in the figure, the PR and PRM model could yield more net revenue than the reactive model when the failure frequency is high. In particular, the proactive-reactive model yields 26.8% net revenue improvement over the reactive model when setting $\theta = 0.1$. And an average

improvement of 11.5% is achieved when $\theta \leq 0.5$. This suggests that the proactive-reactive model makes more sense in unreliable systems. Furthermore, the proactive-reactive model also outperforms the reactive model in rather reliable systems where $\theta > 0.5$.

Figure 6(c) shows that the PR and PRM model yields a greater net revenue than the reactive model under different P_{SLA} values. Net revenue yielded by the proactive-reactive model does not decline much because most penalty costs are avoided by possible proactive recovery and VM migrations. Whereas the reactive model has to pay the penalty when failure occurs, and penalty cost increases as P_{SLA} increases.

Figure 6(d) shows the net revenue under different levels of false-negative ratio ranging from 0.05 to 0.7. With the increase of false-negative error, there is a slight decrease on the net revenue by the PR and PRM model, whereas the reactive model fluctuates around a certain level because the reactive model does not employ failure prediction and the fluctuation is due to the random values used in the experiments. Our proactive-reactive model averagely yields 3.2% improvement on the net revenue over the reactive model, and performs similarly with reactive model when $fn \geq 0.7$.

Figure 6(e) shows the impact on net revenue from the false-positive ratio under a fixed $fn = 0.25$. Net revenue obtained from PR and PRM model decreases gradually over the false positive ratio. Moreover, the differences on revenue between proactive-reactive and reactive model decreases as the false-positive ratio increases. This is because a high false-positive ratio results in a large number of meaningless migrations, which come with migration costs. Figure 6(d) and Figure 6(e) suggest that our algorithm still performs well with even modest prediction accuracy (i.e., $fn \geq 0.5$ or $fp \geq 0.5$).

6 Conclusions

In this paper, we address the problem of cost-efficient fault management, and present revenue driven server management and scheduling algorithm for cloud systems. Using the algorithms, cloud providers could obtain a significant improvement on net revenue when serving the same jobs. In particular, our proposal could yield at most 26.8%, on average 11.5% net revenue improvement when the failure frequency is high. In the future, we will study more scheduling algorithms working together with the proposed server management model. Our goal is to maximize the net revenue for cloud providers without affecting the performance.

References

1. Bobroff, N., Kochut, A., Beaty, K.: Dynamic Placement of Virtual Machines for Managing SLA Violations. In: 10th IFIP/IEEE International Symposium on Integrated Network Management, pp. 119–128 (2007)
2. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. In: DSN 2006, pp. 249–258 (2006)
3. Hoelzle, U., Barroso, L.A.: The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, 1st edn. Morgan and Claypool Publishers (2009)
4. Dean, J.: Experiences with mapreduce, an abstraction for large-scale computation. In: PACT 2006, pp. 1–6. ACM (2006)

5. Vishwanath, K.V., Nagappan, N.: Characterizing cloud computing hardware reliability. In: SoCC 2010, pp. 193–204 (2010)
6. Nightingale, E.B., Douceur, J.R., Orgovan, V.: Cycles, cells and platters: an empirical analysisof hardware failures on a million consumer pcs. In: EuroSys 2011, pp. 343–356. ACM (2011)
7. Javadi, B., Kondo, D., Vincent, J.M., Anderson, D.P.: Discovering statistical models of availability in large distributed systems: An empirical study of seti@home. IEEE Transactions on Parallel and Distributed Systems 22, 1896–1903 (2011)
8. Fu, S., Xu, C.Z.: Exploring event correlation for failure prediction in coalitions of clusters. In: SC 2007, pp. 41:1–41:12. ACM (2007)
9. Pinheiro, E., Weber, W.D., Barroso, L.A.: Failure trends in a large disk drive population. In: FAST 2007, pp. 17–28 (2007)
10. Salfner, F., Lenk, M., Malek, M.: A survey of online failure prediction methods. ACM Comput. Surv. 42, 10:1–10:42 (2010)
11. Koomey, J., Brill, K., Turner, P., et al.: A simple model for determining true total cost of ownership for data centers. Uptime institute white paper (2007)
12. Patel, C.D., Shah, A.J.: A simple model for determining true total cost of ownership for data centers. Hewlett-Packard Development Company report HPL-2005-107 (2005)
13. Fitó, J.O., Presa, I.G., Guitart, J.: Sla-driven elastic cloud hosting provider. In: PDP 2010, pp. 111–118 (2010)
14. Macías, M., Rana, O., Smith, G., Guitart, J., Torres, J.: Maximizing revenue in grid markets using an economically enhanced resource manager. Concurrency and Computation Practice and Experience 22, 1990–2011 (2010)
15. Mazzucco, M., Dyachuk, D., Deters, R.: Maximizing cloud providers' revenues via energy aware allocation policies. In: IEEE CLOUD 2010, pp. 131–138 (2010)
16. Mazzucco, M., Dyachuk, D., Dikaiakos, M.: Profit-aware server allocation for green internet services. In: MASCOTS 2010, pp. 277–284 (2010)
17. Abraham, A., Grosan, C.: Genetic programming approach for fault modeling of electronic hardware. In: The 2005 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1563–1569 (2005)
18. Marbukh, V., Mills, K.: Demand pricing & resource allocation in market-based compute grids: A model and initial results. In: ICN 2008, pp. 752–757 (2008)
19. Zheng, Q., Veeravalli, B.: Utilization-based pricing for power management and profit optimization in data centers. JPDC 72, 27–34 (2012)
20. Macías, M., Guitart, J.: A genetic model for pricing in cloud computing markets. In: SAC 2011, pp. 113–118. ACM, New York (2011)
21. Mastroianni, C., Meo, M., Papuzzo, G.: Self-economy in cloud data centers: statistical assignment and migration of vms. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011, Part I. LNCS, vol. 6852, pp. 407–418. Springer, Heidelberg (2011)
22. Rackspace (2012), `http://www.rackspace.com` (Online; accessed January 31, 2012)
23. Lewis, P.A.: A branching poisson process model for the analysis of computer failure patterns. Journal of the Royal Statistical Society, Series B 26, 398–456 (1964)
24. IBM: Ibm system x 71451ru entry-level server (2012), `http://www.amazon.com/System-71451RU-Entry-level-Server-E7520/dp/B003U772W4`