Availability Knob: Flexible User-Defined Availability in the Cloud

Mohammad Shahrad David Wentzlaff
Princeton University
{mshahrad,wentzlaf}@princeton.edu

Abstract

Failure is inevitable in cloud environments. Finding the root cause of a failure can be very complex or at times nearly impossible. Different cloud customers have varying availability demands as well as a diverse willingness to pay for availability. In contrast to existing solutions that try to provide higher and higher availability in the cloud, we propose the Availability Knob (AK). AK provides flexible, user-defined, availability in IaaS clouds, allowing the IaaS cloud customer to express their desire for availability to the cloud provider. Complementary to existing high-reliability solutions and not requiring hardware changes, AK enables more efficient markets. This leads to reduced provider costs, increased provider profit, and improved user satisfaction when compared to an IaaS cloud with no ability to convey availability needs. We leverage game theory to derive incentive compatible pricing, which not only enables AK to function with no knowledge of the root cause of failure but also function under adversarial situations where users deliberately cause downtime. We develop a high-level stochastic simulator to test AK in large-scale IaaS clouds over long time periods. We also prototype AK in OpenStack to explore availability-API tradeoffs and to provide a grounded, real-world, implementation. Our results show that deploying AK leads to more than 10% cost reduction for providers and improves user satisfaction. It also enables providers to set variable profit margins based on the risk of not meeting availability guarantees and the disparity in availability supply/demand. Variable profit margins enable cloud providers to improve their profit by as much as 20%.

Keywords flexible availability, cloud availability, failure-aware scheduling, cloud economics, SLA

SoCC '16, October 05 - 07, 2016, Santa Clara, CA, USA.
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4525-5/16/10. . . $15.00.
DOI: http://dx.doi.org/10.1145/2987550.2987556

Categories and Subject Descriptors C.4 [*Computer Systems Organization*]: PERFORMANCE OF SYSTEMS—Reliability, availability, and serviceability; K.6.0 [*MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS*]: General—Economics

1. Introduction

Understanding and overcoming failures in computing systems has always been an arduous challenge. The emergence of billion-transistor designs along with smaller and less reliable transistors have led to an increasing rate of hardware failures. Nevertheless, with all of its complexities, hardware is only one source of failure. The entire computing stack, including hardware and software, has faced tremendous growth in both scale and complexity, increasing the probability that failure occurs in such large systems. Moreover, failure analysis in large-scale computing systems has become extremely complicated due to the deployment of hundreds of thousands of machines with complex interactions.

Many researchers have studied failure along with its sources and consequences in large-scale computing infrastructures [41, 57, 59] as well as cloud environments [19, 22, 65]. Failure in the cloud can have major negative consequences such as propagated service disruptions [66], considerable energy waste [31], and more importantly negative effects on provider's reputation [33]. Cloud providers have used many effective techniques to increase the reliability of their infrastructures, but to date, failures are still frequent.

One of the main concerns of cloud customers is availability. According to Pan et al. [50], 73% of customer/provider service level agreement (SLA) negotiations included availability concerns. Availability is considered to be both the number one obstacle to the growth of cloud computing [16] and the most important opportunity for cloud providers [53]. At the same time, surveys [35] illustrate that different customers have different downtime demands, depending on their application. For instance, a low-end, non-critical web hosting service neither has the same availability demand nor the ability to pay for availability that a mission-critical banking service has.

Infrastructure as a service (IaaS) cloud providers generally use a diverse set of hardware components in their data centers. Different generations of one Intel processor family have different reliability, availability, and serviceability (RAS) features [11]. Undoubtedly, providers like Amazon Web Services that use different families and generations of Intel processors [3] have considerable reliability heterogeneity. Moreover, running different operating systems on the same processor can lead to various downtimes [11]. Likewise, different IaaS components, such as memory DIMMs, disks, network switches, power supplies, etc., can also result in varied reliability.

The inevitability of failures, different market demands for availability based on application and IaaS customer needs, and the heterogeneous nature of component reliability, all require a shift from the conventional approach of maximizing availability. Like many researchers have alluded to, there is inefficiency in using fixed availability service level objectives (SLOs) [60, 63], and we believe that both providers and customers can benefit from flexible availability.

In this paper, we propose the Availability Knob (AK) for IaaS clouds. AK enables cloud providers to serve customers with various availability demands. On the customer side, AK allows customers to express their true availability needs and be charged accordingly. Providers benefit from the economic advantages of such flexibility and customers need only to pay for the minimum availability they require. We explore the implications of flexible availability on SLAs and derive economic incentives to prevent customers from gaming the system or providers from deliberately violating SLOs. We then explore and discuss how AK can make more profit for cloud providers. In order to evaluate AK and study different design trade-offs, we developed a stochastic cloud simulator, which enables simulating large-scale infrastructure for extended periods of time; something necessary for our failure-related system. We also implement an AK prototype using the OpenStack platform. In our evaluation, we show that AK has the potential to reduce the cost for the cloud provider, increase provider profit, and improve user satisfaction (meeting user availability needs).

2. The Availability Knob

In contrast to typical IaaS clouds that only offer a fixed availability guarantee to customers, the Availability Knob (AK) allows IaaS cloud customers to request their desired availability objectives and be charged accordingly. AK permits a provider to employ the failure history of its infrastructure to wisely serve different availability demands. It also allows providers to exploit the usual heterogeneity of components in a profitable manner. Figure 1 conceptually depicts how AK enables customers to address their availability demands and schedules them based on machine reliability.

In this section, we discuss the two IaaS elements that should be changed to accomplish AK: service level agreements (SLAs) and the scheduler. We also mention the existing availability monitoring tools/techniques which can be utilized in Section 2.3.

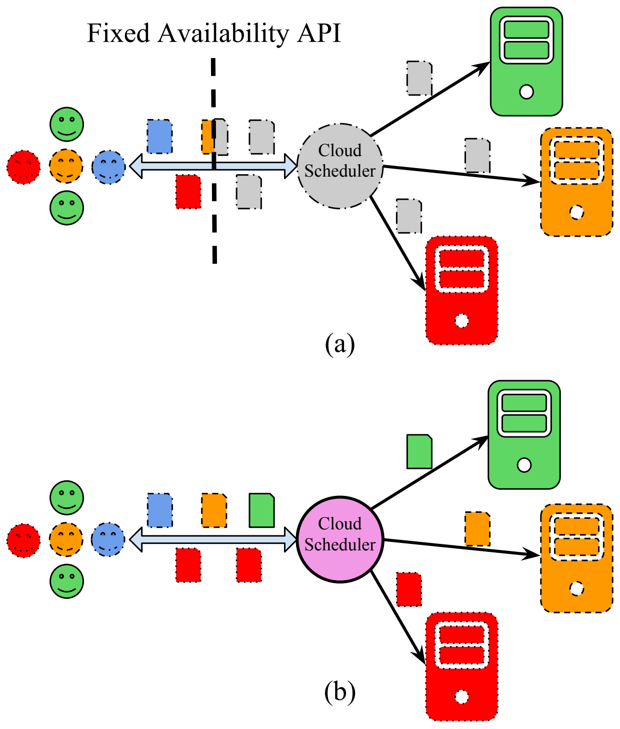


Figure 1. Top (a) shows how availability needs are not conveyed to the cloud scheduler. Bottom (b) shows how AK enables communicating availability information and can schedule based on machine reliability. Different colors represent different availability/reliability.

2.1 SLA for Flexible Availability

Availability is a crucial metric for quality of service and is included in almost all cloud SLAs. While availability is generally defined as the uptime of a service in a specific time period, many assumptions can affect the way it is calculated. For instance, different cloud providers use different service guarantee granularities, guarantee exclusions, time granularities, and measurement periods to measure the availability [18]. According to a comparison [67] of public cloud SLAs, a monthly measurement period is the most common period, service granularities range from a single instance (VM) to all of a customer's VMs in multiple availability zones (Amazon EC2), and time granularities differ from a minute to an hour. The way providers track availability can also be different. For instance, a service provider was reported to track only the internal system availability rather than user accessibility [46]. Moreover, each provider excludes certain events from availability measurement. For example, under seven conditions, Amazon EC2 excludes any downtime from service commitment calculations [2],

Parameter	Description
a	Delivered availability
A	Service commitment (Requested availability)
α	Period portion where A is required.
C	Cost
Ci	Initial cost (not including penalty)
D	Demand
DT	Downtime
DTF	Downtime fulfillment
γ	Exponent in WTP
i	Period index
j	User index
λ	Base in WTP
m	Profit margin (%)
M	Absolute profit margin
P	Price
R	Risk
S	Service
sc	Service credit (%)
SC	Absolute service credit
t	Time
T	Availability period starting time
U	Set of all users
ω	Coefficient in WTP
WTP	Willingness to pay

Table 1. List of parameters used in the paper.

or Google Compute Engine's (GCE) SLA does not apply to many problems including "errors caused by factors outside Google's reasonable control" [5].

Apart from the differences in availability definitions and measurements, service credit (penalty) is also often calculated in different manners. While providers like Amazon EC2, Google GCE, and Microsoft Azure use non-linear service credit functions, some providers use linear penalty schemes [67]. Commonly, the onus is on the customer to report outages and provide the cloud provider with all the necessary information to validate the claim, often within a limited time period (e.g. two month period for Microsoft Azure [6]).

Lack of clarity is an undeniable consequence of such differences in commitment/penalty calculations for cloud customers. Enabling end-users to select their desired uptime and measurement period results in a more sensible notion of availability that allows them to run VM instances tailored to their applications' availability requirements.

Despite existing SLAs that are designed to serve fixed availability commitments, the Availability Knob offers SLAs with configurable availability commitments and measurement periods. Therefore, the service credit function (penalty function) would not be identical for different availability demands. Although the service credit function is determined by each provider based on its own strategies and redefining it is not in the scope of our study, we discuss how a given service credit function can be adjusted to fairly serve different customers with different availability requirements.

Let's assume that a provider is offering a service credit function of $sc_A(a)$ to its clients, where a is the delivered

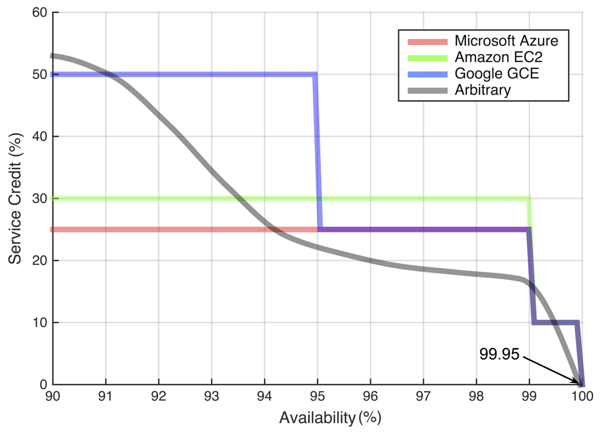


Figure 2. Comparison of four service credit functions.

availability (uptime percentage) and A is the service commitment value (Table 1 describes all the parameters used in the paper). Usually, a service credit is paid if the commitment is not met:

$$A = \min(a)\ s.t.\ sc_A(a) = 0. \qquad (1)$$

The service credit function for three public providers as well as an arbitrary service credit function are shown in Figure 2, where service commitment (A) is 99.95%.

One way to adjust an sc function for different clients is to keep them relatively satisfied. For instance, Alice and Bob asking for 5 and 10 minutes of monthly downtime, respectively, should be given back the same service credit when experiencing 6 and 12 minutes of downtime. To do this, we define the downtime fulfillment (DTF) as

$$DTF = \frac{DT_{SLO} - DT_{Delivered}}{DT_{SLO}} = \frac{(a - A)}{(1 - A)}, \qquad (2)$$

where DT_{SLO} is downtime objective and $DT_{Delivered}$ is the customer's experienced downtime during a specific period. In order for customers with different availability demands to have the same level of downtime fulfillment the following should be satisfied:

$$DTF_{A_1} = DTF_{A_2} \Rightarrow \frac{(a_1 - A_1)}{1 - A_1} = \frac{(a_2 - A_2)}{1 - A_2} \qquad (3)$$

As a result, the service credit function, $sc_{A_2}(a)$ can be calculated from $sc_{A_1}(a)$ as

$$sc_{A_2}(a) = sc_{A_1}(ka + (1 - k)),\ k = \frac{(1 - A_1)}{(1 - A_2)} \qquad (4)$$

Figure 3(a) shows the scaling of the arbitrary sc function shown in Figure 2 for some different availability commitment values. Note that other adjustments might be adopted depending on provider's strategies. For instance, the result of scaling with $k' = \sqrt{k}$ is shown in Figure 3(b).

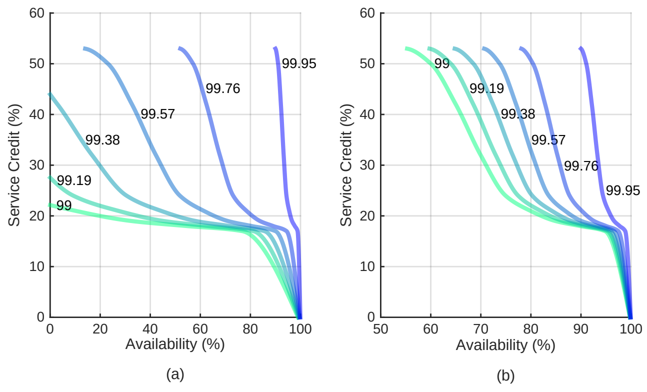


Figure 3. Scaling of a reference service credit function for different availability commitment values.

2.2 Scheduler

AK's VM scheduler aims to maximize the expected provider profit margin while meeting the most SLOs. This means that it not only tries to assign VMs to the cheapest available resources, but it also considers the risk of failure and its resulting service credit (penalty) when doing VM assignment. The Availability Knob scheduler tracks previous machine failures and determines the probability of failure for each specific physical machine (PM) by comparing the time which has passed since its last failure and compares it to the expected mean time between failures (MTBF) for that type of PM. For each VM placement, the scheduler calculates the failure probability as well as expected downtime due to failure and considers a user's delivered/requested availability to calculate the expected overhead cost.

2.2.1 Benign VM Migration (BVM)

The AK scheduler can periodically migrate over-served VMs to cheaper machines, if available. This ability, called Benign VM Migration (BVM) throughout the paper, is a unique consequence of having flexible availability. Providers can deliberately migrate users' VMs in order to lower their operational expenses (OpEx). BVM candidates are selected based on how well they are being served in the current measurement period, using the downtime fulfillment metric (DTF) defined in Equation (2).

2.2.2 Deliberate Downtime (DDT)

Suppose that a customer's delivered availability is going to be higher than the requested availability demand, a provider can deliberately make them experience downtime. Seemingly brutal, such Deliberate Downtimes (DDT) can be used by providers for different strategic purposes such as creating market incentives (for instance to encourage customers to purchase their true needs), better serving needy customers, or lowering energy consumption. The reclaimed resources due to DDT deployment can be used for extra bidding or computational sprinting [70] purposes.

Each user has a maximum affordable downtime, which can be written as

$$DT_{Max}(t) = \max\left(DT_{SLO} - DT(i), 0\right), \;\; t \in [T_i, T_{i+1}). \quad (5)$$

Here t is the time, T_i is the starting time of user's i'th period, and $DT(i)$ denotes the experienced downtime in the i'th period. Now, if the remaining time in the period is less than the affordable downtime ($T_{i+1} - t < DT_{Max}(t)$), the cloud provider can attempt to pause or suspend some or all of the user's VMs.

Each deliberate downtime event contains uncertain VM deactivation and reactivation intervals. In specific, a long reactivation interval can possibly extend the downtime into the next measurement period. This can cause an SLO violation if a user's demand squeezes in the next period. Our experiments show that setting a small safety margin (5% of DT_{Max}) on DDT length can significantly alleviate such violations. In Section 6 we demonstrate the results of applying DDT.

2.3 Availability Monitoring

In order for AK to operate, failure occurrences need to be collected. Due to its wide applicability, performance and availability monitoring has been studied by researchers for a long time [14, 32]. Moreover, there are many industrial solutions, such as Nagios [7], Ganglia [4], and Zabbix [9], available for monitoring cloud availability. Systems like Flex-ACMS [27] even enable automatic deployment of multiple cloud monitoring tools. Therefore, the existing availability monitoring solutions are adequate for the AK to use.

2.4 Deployment of AK

Unlike IaaS ideas introducing novel architectures [17], no hardware change is required to deploy the AK. This reduces technology adoption costs for cloud providers. On the other hand, AK can be used in IaaS clouds as an optional feature. Conventional fixed availability is a subset of AK, and this allows customers to continue using conventional fixed availability services.

3. Economics of the Availability Knob

The foundation of the AK can be summarized in a single sentence: the higher the delivered availability, the more it costs the cloud provider. Availability not only directly relates to the resource usage percentage, but also providing high availability (HA) requires excess system costs. Any HA solution, such as replication, checkpointing, or machine resilience, introduces CapEx[1]/OpEx overheads.

Allowing adaptation to various availability commitments, the Availability Knob offers product flexibility. This leads to supply chain flexibility, a strategically important capability [61]. In this section, we explore the price incentives needed to prevent customers from gaming the system and to

[1] Capital Expenditure

	Defective Client Request higher availability and cause DT	Healthy Client Request the desired availability
Responsive Provider	$(M_{A_2} - SC_{A_1}(A_2), P_{A_1} - SC_{A_1}(A_2))$	(M_{A_2}, P_{A_2})
Lazy Provider introducing excess DT	$(M_{A_3} - SC_{A_1}(A_3), P_{A_1} - SC_{A_1}(A_3))$	$(M_{A_3} - SC_{A_2}(A_3), P_{A_2} - SC_{A_2}(A_3))$

Table 2. Payoff matrix showing provider margin and customer price for different causes of downtime.

prevent cloud providers from being deliberately negligent. Moreover, we address how AK can lead to higher profits for cloud providers.

3.1 Pricing for Incentive Compatibility

As many studies [29, 49, 57] have shown, software is one of the root causes of failure in computer systems. By virtue of VM isolation in the cloud, failures in a single VM do not affect other co-tenants (unless the hypervisor suffers a failure) [45]. However, such possible software failures can affect the availability of the VM itself. Often, it is difficult to distinguish software-induced failures from failures caused by hardware (e.g. soft error incident), etc.

We believe, similar to other researchers [39], that downtime measurements should be inclusive of all classes of failure, including software failure. We further argue that instead of adopting measures to investigate the cause of a downtime incident, which can impose performance overheads and might not be accurate, IaaS providers can use price as a more effective tool. We use game theory to show how prices can be set in such a way that clients who are running defective software or are deliberately causing software failures pay commensurately higher prices. Likewise, providers are incentivized to meet their service objectives.

3.1.1 Healthy Client, Responsive Provider

As shown in Table 2, a payoff matrix can be formed to find the Nash Equilibrium for different provider/client behavior scenarios. On the vertical axis, a provider can either provide the availability requirements or introduce some extra downtime due to lazy management. Across the horizontal axis, a client can either run defective software which reduces the availability from A_1 to A_2, resulting in some service credit return, or instead run healthy, reliable, software and originally ask for the lower A_2. In this game theoretical representation, maximizing the absolute profit margin (*M*) is the primary objective of a provider, while the client tries to minimize the price (*P*). Here, *SC* represents the absolute service credit ($SC_A(a) = sc_A(a) \times P_A$). Now, meeting the following constraints of:

$$\forall A_1, A_2, A_3 \; s.t. \; A_3 \leq A_2 \leq A_1 :$$

$$1. \; M_{A_3} - M_{A_2} \leq SC_{A_1}(A_3) - SC_{A_1}(A_2) \qquad (6a)$$

$$2. \; SC_{A_1}(A_3) - SC_{A_2}(A_3) \leq P_{A_1} - P_{A_2} \qquad (6b)$$

ensures that there exists an equilibrium (highlighted in gray in Table 2), where the provider maximizes its profit margin by doing its best to meet the SLA and the client pays minimally by asking for its true availability demands. Constraint (6a) assures that when providing less availability, a provider's margin increase is less than the increased penalty. Therefore, it guarantees a provider's adherence to the availability SLO to maximize its margin. At the same time, (6b) ensures that a customer lowering the availability to gain service credits would still be more expensive than initially asking for the desired availability.

Cloud providers can enforce (6a) and (6b) by setting their pricing schemes properly. Clients, on the other hand, should be able to verify these constraints. Even though *P* and *sc* (and as a result, *SC*) functions are known to both parties through the SLA, the *M* function is conventionally not revealed to clients. While the cloud client may not exactly know *M*, it can typically approximate this function. If the client believes that (6a) is not maintained, it can choose not to use the cloud provider. Also making this easier for the client is that $M_{A_3} - M_{A_2}$ is typically negative because margins are typically higher for harder-to-provide availability values.

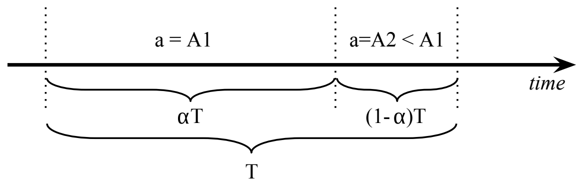


Figure 4. The aggregate availability requirement of a client in a measurement period (T).

3.1.2 Incentive Compatibility for Variable Demands

Consider a client who does not always require the initial availability, A_1. But rather, sometimes can afford a lower availability, A_2. Figure 4 shows the aggregate availability requirement of such a client in a measurement period (T), where α is the portion that high availability A_1 is required. This client has three choices:

1. Ask for A_1 during the whole period and pay the cost overhead. This strategy leads to the total price of P_{A_1}.
2. When the high availability is not required, deliberately crash its VM(s) to lower the availability to A_2 and earn service credits. As a result, the total price paid in a period would be

$$P_{A_1} - SC_{A_1}(\alpha A_1 + (1-\alpha)A_2). \qquad (7)$$

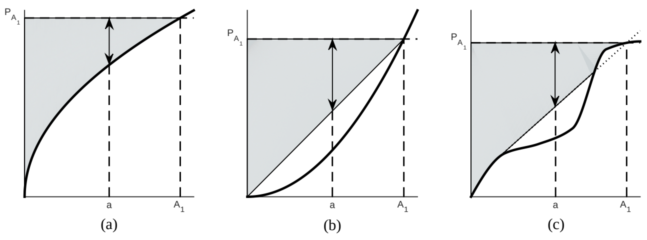


Figure 5. Upper bound of SC (gray area) and max $(SC_{A_1}(a))$ (double-arrow) for (a) a concave, (b) a convex, and (c) an arbitrary price function.

3. Change its availability need to A_2 and back to A_1, whenever needed. This way the client pays

$$\alpha P_{A_1} + (1-\alpha)P_{A_2}. \tag{8}$$

While option 1 is the most expensive, in order to motivate the client to choose option 3 over 2,

$$\alpha P_{A_1} + (1-\alpha)P_{A_2} < P_{A_1} - SC_{A_1}(\alpha A_1 + (1-\alpha)A_2) \tag{9}$$

should be satisfied. This relation can be rewritten as

$$\begin{gathered} \forall \alpha \in [0,1],\ \forall A_2\ s.t.\ A_2 < A_1 \\ SC_{A_1}(\alpha A_1 + (1-\alpha)A_2) < (1-\alpha)(P_{A_1} - P_{A_2}). \end{gathered} \tag{10}$$

Equation (10) correlates the service credit function (SC) with the price function (P) to maintain a client's economic incentive. Given a price function, this equation sets the upper limit on SC. The geometrical interpretation of (10) is that $SC_{A_1}(a)$ should be bound between the P_{A_1} line and all secant lines drawn between the fixed (A_1, P_{A_1}) point and any variable point on the price function. Figure 5 shows this bound (gray area) for a concave, convex, and arbitrary-shaped price function. For a concave function (Figure 5(a)) that all such secant lines lie under the price function, the SC function is simply bound between P_{A_1} and the price function. By choosing the SC in this bound, a provider implements economic incentives for its clients to request their true demands.

3.2 How does AK make money?

AK helps cloud providers achieve higher profit margins in two major ways. First and foremost is by adapting delivered services to customers' real demands. As discussed in [71, 72], such an ability leads to **higher market efficiency**, where both the cloud provider and cloud customers gain benefit. In Section 6 we show how AK improves user satisfaction, which is essential for maintaining long-term revenue [23]. Second, is by **efficient resource utilization**. AK enables techniques such as BVM and DDT that lower the OpEx and conserve resources. Providers can make additional revenues by bidding reclaimed resources (similar to AWS spot instances) or by offering better computational sprinting.

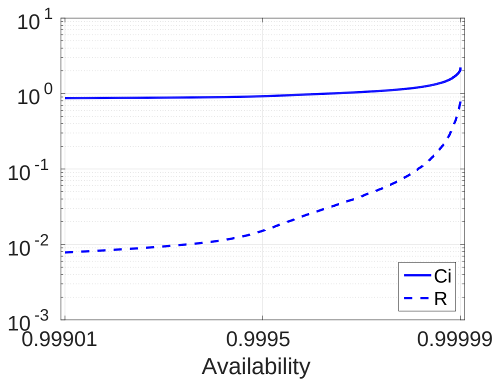


Figure 6. Arbitrary initial provider cost (C_i) and risk (R) functions.

To demonstrate the importance of flexible availability, we show the potential of setting **variable profit margins** based on different availability demands. This enables providers to be compensated for the risk of providing high availability and adjust the price to market demand. Let's assume that 20,000 users are being served in a cloud environment. These AK users have uniform, normal, or bimodal availability demands in the $[99.901\%, 99.999\%]$ range[2]. Consider $Ci(a)$ denoting the initial provider cost function (cost not considering possible penalties), and $R(a)$ expressing the risk function, which determines the risk of not meeting the availability SLO. The expected *absolute profit margin* (M) can be calculated as

$$\begin{aligned} E[M] &= \sum_{j\in U} [(1-R_{A_j})(m_{A_j}Ci_{A_j}) + R_{A_j}(m_{A_j}Ci_{A_j}(1-p))] \\ &= \sum_{j\in U} m_{A_j}Ci_{A_j}(1-pR_{A_j}), \end{aligned} \tag{11}$$

where U is the set of all users, m_{A_j} is a provider's profit margin for the availability requested by user j (e.g. 20%), and p is the penalty (e.g. 10% assuming fixed sc). Considering $Ci(a)$ and $R(a)$ to be as shown in Figure 6, we are interested to know if the provider can increase its overall profit by tuning the profit margin for different availability values. For the sake of fair comparison, we keep the average of the tuned margins to be the same as the flat margin of 20%. Figure 7 depicts the availability demand distribution, new modified margins, and the overall gain over having a fixed flat margin of 20% for all availability needs. The overall gain is a function of risk as well as the service credit function, sc, which for simplicity is assumed to be a fixed value (p) for all users. As seen, for this particular cost function and depending on the demand distribution, risk of service, and service credit, the provider can gain between 1% to 20% higher overall

[2] Throughout this paper, for availability range $[A_1, A_2]$, normal distribution parameters are $\mu = \frac{A_1+A_2}{2}$ and $\sigma = \frac{\mu}{4}$. Bimodal distribution is created by first forming a normal distribution ($\mu = A_1, \sigma = \frac{\mu}{4}$) and then shifting the left half to A_2. Availability demands greater than 1 ($A > 1$) are neglected.

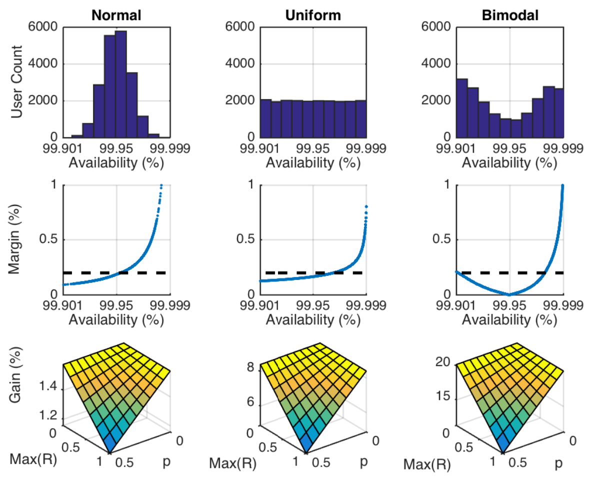


Figure 7. Gaining more overall profits by tuning profit margins based on user demand and risk of service.

profit by tuning the margins for individual users. Such tuning, which is based on **risk of service** and **supply/demand**, brings availability-specific pricing flexibility to providers.

4. Prototype Implementation

Similar to other researchers that have used OpenStack for their cloud availability studies [34, 37, 40], we implemented AK's prototype by integrating it into OpenStack [8]. We modified OpenStack's dashboard (Horizon) and compute (Nova) components to add the notion of availability to associated APIs and databases (DBs) as well as the scheduler. As Table 3 shows, the integration of AK requires only minor modification to OpenStack Nova database. APIs between the Horizon and Nova components as well as internal Nova APIs that connect to the Nova DB have also been modified.

Figure 8 shows the modified dashboard, where the *admin* can manage availability options. Each availability option, has an availability value (**avail_val**), calculation period (**period**), and price factor (**price_factor**). For instance, an option can have 99.99% availability, calculated over a 30-day period, and a price factor of 1.12 (12% extra price compared to the default 99.95%). Each virtual machine instance has an availability option tied to it, which is selected at the launch time and can be modified later.

The OpenStack scheduler is composed of different filters that perform filtering and weighting of available hosts (physical nodes) to schedule a VM instance [44]. We added a new weigher as well as a filter to weight and select suitable hosts depending on an instance's availability demand. This is done by comparing the time which has passed since the last failure of a node to its MTBF. We modified OpenStack's Libvirt driver to track availability of compute nodes and update failure records when any status changes occur. In future, we

Table	New Table	Added Field
compute_nodes	✗	last_failure lastlast_failure **mtbf** **mttr**
instances	✗	availability_op_id avail_val period price_factor
availability_ops	✓	**avail_val** **period** **price_factor** + 9 general fields
availability_op_projects	✓	availability_op_id + 6 general fields

Table 3. Summary of OpenStack Nova DB modifications.

plan to expand the scope of AK to OpenStack's block storage (Cinder) and image service (Glance) components, as well.

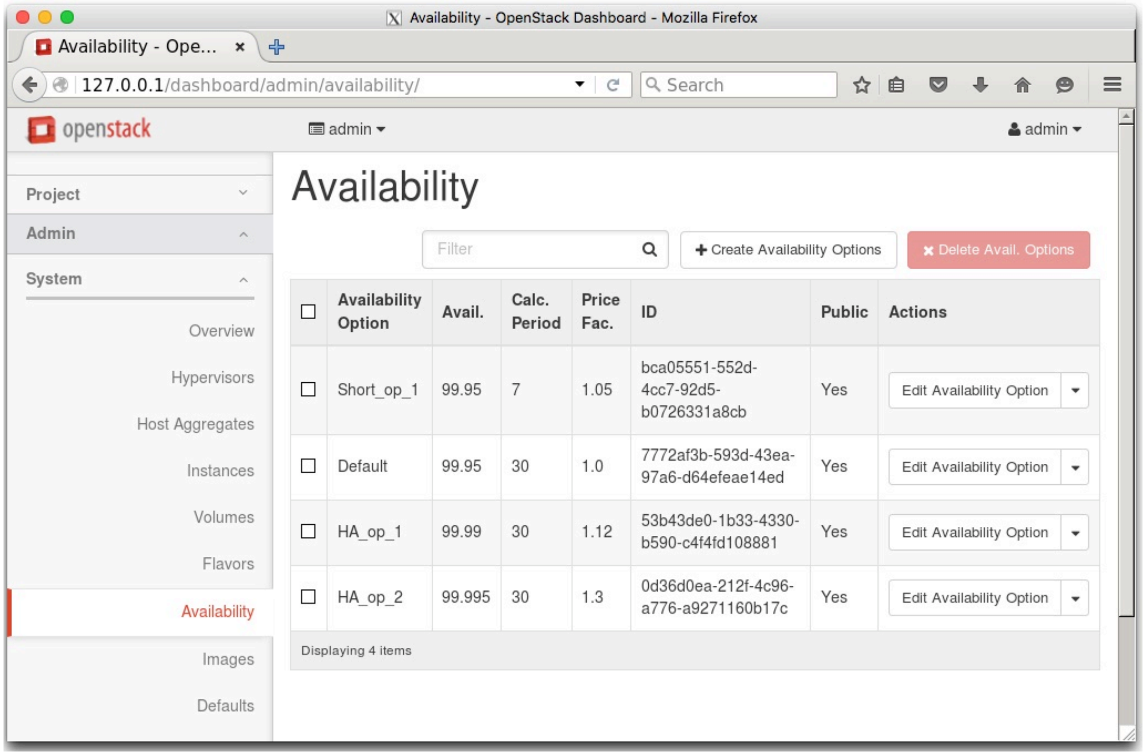
In our evaluation section, we focus on using a stochastic simulator over using our OpenStack prototype because it enables us to vary machine types which would be impractical to do with a small cluster. Also, faults are infrequent enough that it would be difficult to get a statistically significant number of errors in our cluster over a short period of time. Therefore we use AKSim to evaluate large data centers with many different types of machines over long, simulated, periods of time.

5. AKSim: The Stochastic Cloud Simulator

Due to the nature of failures being very uncommon in modern computing systems, large-scale/long-term analysis of AK is necessary; especially considering the fact that accelerated testing is extremely inaccurate for complex systems (e.g. a data center). Therefore, although we built the AK prototype by integrating our idea into OpenStack, we believe that the best way to evaluate AK is through simulation of large infrastructures over long periods of time. To do so, we developed a high-level stochastic cloud simulator (AKSim) in MATLAB.

We should elaborate that using available workload traces, such as Google's cluster workload traces [54], does not suit our study, since such traces have no machine reliability information. This prevents us from investigating the impact of component heterogeneity. Also, we study environments with diverse availability needs and using the available traces that are for fixed availability environments is futile.

The simulation environment hosts VMs from thousands of users. Each user is randomly selected from nine different application categories, such as web hosting, analytics, email service, etc. Each usage scenario can require different resources, and might run multiple VMs. The VM lifetime and reactivation probabilities are random variables with normal distributions that depend on the user application category. The availability needs of AK users can follow arbitrary distributions between different availability ranges and



Availability

	Availability Option	Avail.	Calc. Period	Price Fac.	ID	Public	Actions
☐	Short_op_1	99.95	7	1.05	bca05551-552d-4cc7-92d5-b0726331a8cb	Yes	Edit Availability Option
☐	Default	99.95	30	1.0	7772af3b-593d-43ea-97a6-d64efeae14ed	Yes	Edit Availability Option
☐	HA_op_1	99.99	30	1.12	53b43de0-1b33-4330-b590-c4f4fd108881	Yes	Edit Availability Option
☐	HA_op_2	99.995	30	1.3	0d36d0ea-212f-4c96-a776-a9271160b17c	Yes	Edit Availability Option

Displaying 4 items

Figure 8. A screenshot showing the added Availability panel in the AK OpenStack dashboard.

Algorithm 1 Main tasks performed in each AKSim simulation step.

```
1:  Create/end users
2:  loop over all users:
3:      Update costs
4:  loop over all machines:
5:      Reactivate down machines (based on MTTR)
6:      Fail running machines (based on MTBF)
7:  loop over all VMs:
8:      Update users' uptime/downtime
9:      if VM is active then
10:         Expire instance (based on instance lifetime)
11:         Migrate instance (based on host status)
12:         Perform BVM (if applicable)
13:         Perform DDT pause/unpause (if applicable)
14:     else
15:         Reactivate instance (if applicable)
```

we mention it for each test. The computing environment is composed of limited physical machine (PM) types, having different resources and costs as well as different aggregated annualized failure rates (AFRs). Aggregated AFRs incorporate any root cause of failure affecting end user service, such as hardware, software, network, etc. Algorithm 1 shows the main tasks performed in each simulation step of AKSim.

It is worth mentioning that feeding the simulator with precise AFR values is not trivial. The main reason is that such reliability data for real data centers is usually not publicly available. Furthermore, not only does every computing environment have its unique failure rate, but also the differences in the calculation of such availability metrics complicates the use of reported failure data. For instance, Microsoft reports that 9% of machines have been replaced in a 14 month period [65], which translates into $AFR = 8\%$ and $MTBF > 12yrs$. Such a low failure rate is a result of authors only considering machine replacements in AFR calculations. However, Sathiamoorthy et al. [55] mention an average of about 25 failed nodes, **daily**, in a 3000 node production cluster, which is noticeably higher than the previous record. Google's cluster workload traces [54] reveal that 40.9% of nodes have been removed at least once in a 29-day period [24]. For our simulations, we assume the MTBF to be between 12 to 60 months ($AFR \in [18\%, 63\%]$).

Many studies [41, 57, 58] have shown that the time between failure in large computing systems is well modeled by the Weibull distribution. Nevertheless, creating a Weibull distribution using only AFR (or MTBF) is not possible, as the distribution is characterized by two parameters. Also, as discussed by Krasich et al., the mean of the Weibull distribution "*is in no way related to the mean times to the failure occurrences* [42]." Therefore, our simulator uses the MTBF values corresponding to various machine types to first generate intermediary exponential distributions. Subsequently, Weibull distributions are formed by deliberate imprecise fitting to those exponential distributions. Performed using in-adequate data points, such imprecise fitting introduces some noise and makes the distributions different. In general, having access to more information than mere failure rates would allow one to form the Weibull distributions directly. The generated Weibull distributions are used to determine the MTBF of each single PM. After each machine failure, the simulator uses a machine's failure distribution to assigns a new MTBF to it.

We use the Amazon EC2 on-demand instance prices (available through AWS Price List API), instance machine types [3], and AWS pricing principles [12] to reverse engineer the cost for compute, memory, storage, and network. To calculate the cost from the price, we adopt the simplifying assumption of AWS having a fixed 20% profit margin[3]. Machines simulated by AKSim can have different component types with different costs. For instance, simulated machines can use provisioned IOPS, general purpose, and magnetic volume types, similar to Amazon EBS [1], and we charge their tenants respectively.

6. System Evaluation

As mentioned in the previous section, in order to evaluate our system we take advantage of stochastic simulations, which enables evaluating AK in large infrastructures over extended periods of time. In Section 6.1 we demonstrate the performance of AK's availability-aware scheduler. Then in Section 6.2, we show how AK improves users' satisfaction by delivering their desired demands and charging them respectively. The effectiveness of BVM in decreasing costs is discussed in Section 6.3. Finally in Section 6.4, we show how catastrophic events hurt AK customers differently than in a fixed availability regime. Our results show that by utilizing a flexible availability API, an availability-aware scheduler, BVM, and DDT, AK reduces the provider costs up to 10.9% compared to a conventional IaaS environment.

6.1 Availability-aware VM Scheduling

As discussed in Section 2, AK performs availability-aware VM scheduling through anticipating the risk of each PM considering its failure history. Moreover, it performs Benign VM Migrations (BVMs), where over-served VMs are periodically migrated to cheaper resources, when available. Figure 9 compares the requested and delivered monthly availability values, in a data center (DC) with 1,000 machines and 12,000 users, over the course of 6 months. BVM is performed every hour for the top 10% over-served users. Here, user demand has a normal distribution in the $[99.901\%, 99.999\%]$ range. As shown, the scheduler fails to meet the availability SLO in less than 0.02% of cases. The color of data points changing from magenta to blue shows lower DTF values and the bisector (dashed line) represents

[3] According to Forbes [15], AWS had a margin of 16.9% in the first quarter of 2015 and this number was predicted to grow.

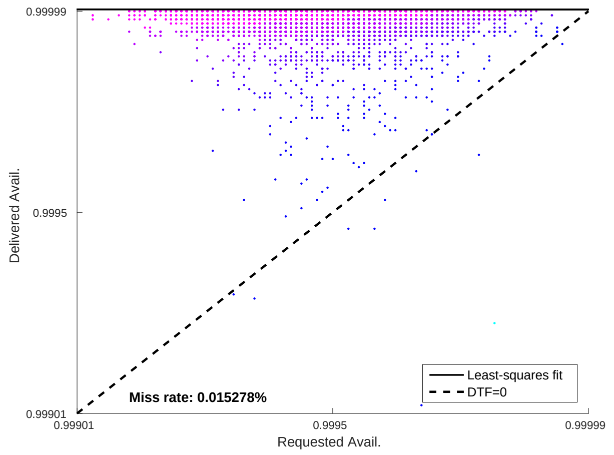


Figure 9. Delivered vs. requested availability in a DC with 12,000 users, where more than 99.98% of SLOs are met.

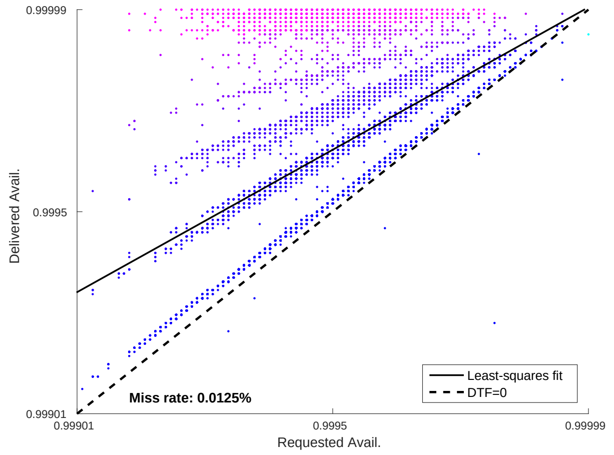


Figure 10. Delivered vs. requested availability in a data center with 12,000 users after applying DDT.

$DTF = 0$. Points below the dashed line represent missed SLOs.

We introduced Deliberate Downtime (DDT) as a tool to shape delivered availabilities, mainly to build market incentives (2.2.2). Having similar simulation parameters as the previous experiment, Figure 10 shows how enabling DDT affects the delivered service. Comparing it with Figure 9, one can see how the average delivered availability is lowered without missing SLOs. Each diagonal distribution in the plot corresponds to users with the same fraction of their VMs being paused at the end of a period, and in particular, dots on the bisector correspond to the case where all VMs of a user experienced DDT.

The scheduler's performance profoundly depends on availability demand distribution. Table 4 depicts the scheduler performance with and without applying DDT for three different demand ranges and various distribution shapes. It can be seen that the SLO miss rate increases for higher/tighter

Range	Dist.	No DDT		DDT		$\Delta C(\%)$
		Miss(%)	DTF	Miss(%)	DTF	
[Two 9's, Four 9's]	N	0	0.999	0	0.300	0.116
	U	0.004	0.997	0.001	0.299	0.263
	B	0.004	0.997	0.007	0.300	0.125
[Three 9's, Five 9's]	N	0.015	0.987	0.013	0.301	-0.021
	U	0.342	0.972	0.424	0.303	0.102
	B	0.458	0.966	0.529	0.305	0.294
[Four 9's, Six 9's]	N	2.572	0.878	2.783	0.344	0.220
	U	4.222	0.835	4.456	0.343	0.074
	B	5.178	0.806	5.382	0.332	0.136

Table 4. The AK scheduler performance w. and w.o. DDT (N: Normal, U: Uniform, B: Bimodal).

availability demand distributions. Furthermore, since most of the misses are for very high availability demands, the higher the concentration of demands at the high reliability tail, the higher the miss rate ($Miss_N < Miss_U < Miss_B$). Table 4 also reflects how performing DDT decreases the average downtime fulfillment (DTF) noticeably, with just a small miss rate increase. It is also worth noting that pausing VMs by DDT results in a minor cost saving, as well (last column, ΔC).

6.2 Increased User Satisfaction

One of the benefits of AK is allowing customers to express their different availability demands and be served accordingly. This, intuitively, would lead to more customer satisfaction. In order to investigate it formally, we define a measure for service satisfaction to address two main service elements: demand satisfaction (D_{sat}) and price satisfaction (P_{sat}). While the former captures how good of an availability was delivered to a customer, the latter compares the actual service price to a customer's desired price. In fact, these two capture perceived quality (PQ) and perceived value (PV) of the service, which is necessary for customer satisfaction indices (CSIs) [62]. More factors, such as ease of deployment, also contribute to user satisfaction, but their precise evaluation necessitates a market study with real customers and is out of the scope of this paper. We use DTF to represent demand satisfaction and assume that the average willingness to pay (WTP) can be represented by

$$WTP = \omega\lambda^{\gamma(A-A_c)} \tag{12}$$

function families, where A_c is the fixed availability value without AK and ω, λ, and γ are shaping parameters. In general, the WTP function can have any arbitrary shape and can only be precisely determined by market analysis after deploying AK. But in Equation (12), we model it by a large family of exponential functions. We assume that users are not willing to pay more for less availability and thus WTP is non-decreasing ($\lambda > 0, \gamma > 0$). Therefore, the

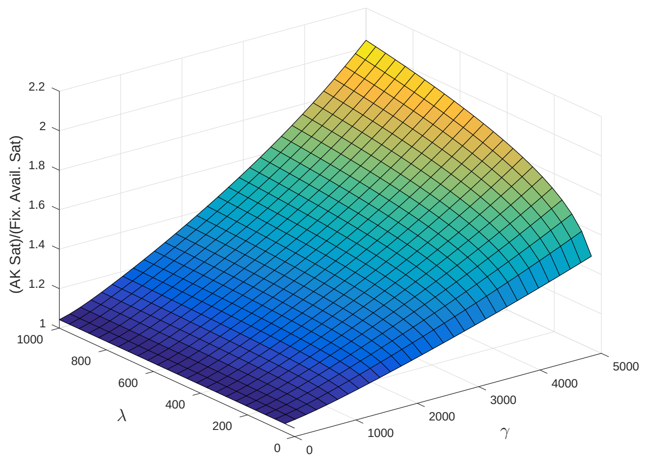


Figure 11. User satisfaction using AK commitment value is always greater than having fixed availability option. λ and γ are shaping parameters of the users' willingness to pay function.

service satisfaction can be written as

$$S_{sat} = D_{sat} \times P_{sat} = DTF \times \frac{WTP}{P_{charged}} = DTF \times \frac{\omega\lambda^{\gamma(A-A_c)}}{P_{charged}}. \tag{13}$$

Figure 11 shows the ratio of S_{sat} with AK over service satisfaction with a fixed A_c commitment value. The delivered availability (A) and charged price ($P_{charged}$) values are fed by our simulation results. Each user's $P_{charged}$ is calculated by taking an initial service cost together with a 20% fixed margin and a 10% service credit in case the SLO is missed. As seen, no matter what the values of λ and γ in (13) are, users would always be more satisfied on average using AK vs. not using AK[4]. It is also worth noting that as users become less willing to pay more for higher reliability ($\gamma \to 0$), the satisfaction improvement offered by AK becomes less significant.

6.3 Benefits of BVM

The concept of BVM was described in Section 2.2.1. To demonstrate how BVM lowers the costs, we simulate a data center with 2,000 machines and 25,000 users having a uniform demand distribution in the $[99.901\%, 99.999\%]$ range. The data center resources have three categories of servers, A, B, and C, with an MTBF of 5, 3, and 1 year(s), respectively. The simulated data center has 200, 600, and 1,200 machines of types A, B, and C. BVM is performed every hour for the top 10% over-served users. Depicting normalized utilization of different machine types during a 30-day period, Figure 12 demonstrates how BVM offloads VMs to cheaper less reliable resources when possible. For this specific example, de-

[4] Result is independent of ω, as it is a common factor of numerator and denominator.

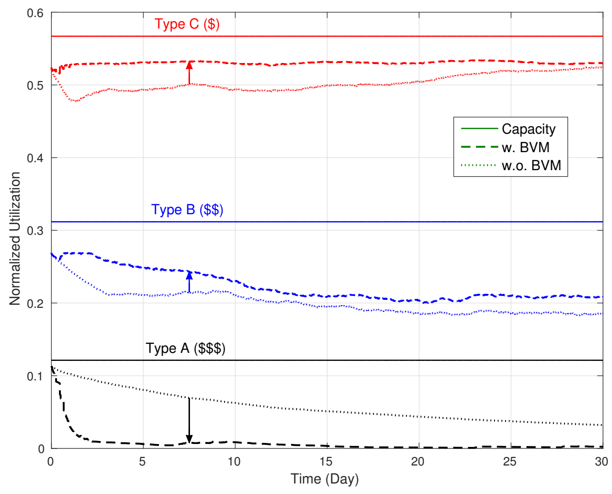


Figure 12. Applying BVM offloads VMs to cheaper less reliable resources when possible. Utilization of each machine type is normalized to its total compute capacity.

$[Type_A, Type_B, Type_C]$	Cost Redu. (%)	Δ Miss Rate (%)
[50,150,800]	5.8	0.60
[50,250,700]	5.3	0.58
[100,300,600]	6.6	0.69
[150,350,500]	5.3	0.65
[200,400,400]	3.6	0.63
[300,400,300]	2.5	0.76
[450,100,450]	2.4	0.68

Table 5. Dependence of BVM cost reduction and induced SLO misses on machine type blend.

spite the small increase in SLO misses (0.34%), BVM leads to a 7.1% cost reduction (considering penalties given back).

Benefits of BVM depend on the resource type blend as well as resource utilization. Table 5 shows how different resource type mixtures lead to various cost reductions and SLO miss rate increases (cost reductions include the increased misses). Figure 13 depicts how resource utilization affects BVM cost savings. Under low utilization (Region 1), only the cheapest resources are mainly utilized and thus cheaper resources are not available for benign migrations. As the number of users increases, machine type B, and eventually type A become utilized (Reg. 2, and 3, respectively) and BVM has more to offer. BVM enjoys the diversity of resources in Reg. 3, but after some point, the data center becomes highly utilized and migrations become more and more difficult. That is the reason why the benefits of BVM diminishes in Reg. 4, where in this example utilization exceeds 80%. It is worth mentioning that resource utilization for cloud data centers ranges from 40 to 70 percent [10], so going into Reg. 4 is practically unlikely.

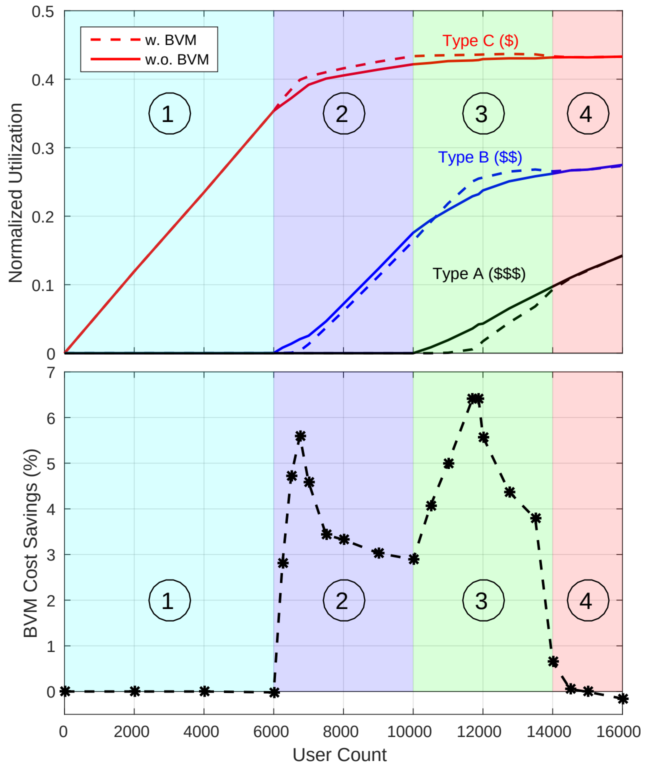


Figure 13. Data center utilization and its impact on BVM cost savings. Under high resource utilization (Region 4), VM migration becomes difficult.

6.4 Catastrophic Failures and AK

Catastrophic failures are downtime events that affect a large number of cloud customers. Such conditions might arise from electric failures, cascading software misconfigurations, natural disasters (e.g. Hurricane Sandy), etc. [22]. AK enables customers to have various availability demands, and thus catastrophic failures have different consequences on its customers, compared to typical fixed-availability regimes.

To compare the impact of a catastrophic failure on AK versus typical fixed-availability environments, we investigate how many SLOs are missed due to such an event in each setting. We simulate two data centers with 3,000 machines and 36,000 users. While users of one have a uniform $[99\%, 99.999\%]$ availability demand range, other's can only ask for fixed 99.5% availability demands. In order to merely evaluate the impact of having a flexible API, we use AK's availability-aware scheduler in both cases. As seen in Figure 14, serving customers with fixed availability results in a sudden rise when the length of failure is more than 3.6 hours, which is the corresponding monthly downtime of 99.5% availability. On the other hand, having various availability demands, the portion of AK customers suffering from the catastrophic event is relative to the event's duration.

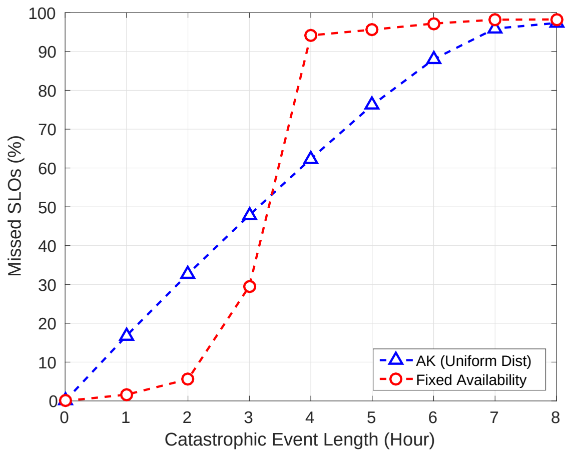


Figure 14. Catastrophic failure event hurts AK SLOs differently compared to the fixed availability regime.

7. Related Work

The majority of studies on availability/reliability of cloud computing systems focus on providing high availability. Fault tolerance through redundancy and replication of VMs [13, 26], hypervisors [68], data [20, 21, 51], and even hardware [36] has been proposed. Checkpointing is another method to provide fault tolerance [56]. We refer the reader to thorough surveys on cloud resiliency techniques [25, 52].

There have been many studies on proactive fault tolerance and failure management. Nagarajan et al. [48] define a transparent, proactive fault tolerance mechanism for MPI applications in HPC clusters, where VM live migration is provoked by health monitoring. Vallee et al. propose proactive VM pausing, unpausing, or migration if a fault has been predicted in computing clusters [64]. Fu [28] proposes a failure-aware reconfigurable distributed virtual machine (RDVM) infrastructure for high-availability computing clusters. The system comprises of a major component, the FailurePredictor, that predicts future failures for each node based on its performance metrics and previous failure events [30]. In all of these works, the goal is to offer or maintain high availability, whereas our system aims to offer different amounts of availability to a wide range of users.

Javadi et al. build statistical models of failure from availability traces of Internet-distributed systems and show how such knowledge can be applied to a resource brokering problem [38]. AK, however, is designed for IaaS clouds and has the advantage of serving customers with different availability needs.

Some studies have addressed the need for flexible availability. Undheim et al. [63] develop their cloud model with a focus on availability and address the need for differentiated SLAs due to the variety of cloud services. Enumerating the future research opportunities to ensure availability in the cloud, Moreno-Vozmediano et al. suggest per-service implementation of availability monitoring and automatic instance re-deployment [47]. Yaqub et al. introduce an optimal SLA negotiation scheme, where availability is also a negotiable parameter [69]. Limrungsi et al. sketch the high-level idea of on-demand reliability, but their work is mostly devoted to modeling and simulation of the proposed checkpointing technique to provide joint reliability maximization [43]. On the contrary, AK is not just a resilience technique or a SLA scheme, but rather a comprehensive model to realize the flexible availability in IaaS environments. We explored AK's economics, implemented its prototype, and thoroughly evaluated different design trade-offs through simulations.

8. Conclusion

In this work, we introduced the Availability Knob (AK) for IaaS clouds. AK enables new flexibility in conveying availability needs between cloud customers and providers. By adding this additional information across the customer/provider interface, cloud providers can price availability more flexibly, schedule to meet the availability needs of a particular cloud customer without wastefully over-delivering, and build a more dynamic marketplace for availability. We show that AK can reduce cloud providers costs by more than 10%, increase provider profit by up to 20%, and improve user satisfaction with the delivered availability. AK has been implemented within the OpenStack framework to demonstrate that it is easy to add it to real IaaS systems. In conclusion, AK is a low-impact interface change and scheduler modification which can enable benefits for both cloud customers and providers without a major overhaul to cloud management software.

Acknowledgments

We thank Kyle Jamieson, Alexey Lavrov, and Prateek Mittal for feedback on this work. This work was partially supported by the NSF under Grants No. CCF-1217553, CCF-1453112, and CCF-1438980, AFOSR under Grant No. FA9550-14-1-0148, and DARPA under Grant No. N66001-14-1-4040. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

[1] Amazon EBS product details. URL https://aws.amazon.com/ebs/details. Accessed: 2016-08-26.

[2] Amazon EC2 service level agreement. URL https://aws.amazon.com/ec2/sla. Accessed: 2016-08-26.

[3] Amazon EC2 instance types. URL https://aws.amazon.com/ec2/instance-types. Accessed: 2016-08-26.

[4] Ganglia monitoring system. URL http://ganglia.sourceforge.net. Accessed: 2016-08-26.

[5] Google compute engine service level agreement. URL https://cloud.google.com/compute/sla. Accessed: 2016-08-26.

[6] SLA for cloud services. URL https://azure.microsoft.com/en-us/support/legal/sla/cloud-services/v1_0. Accessed: 2016-08-26.

[7] Nagios official website. URL https://www.nagios.com. Accessed: 2016-08-26.

[8] OpenStack official website. URL http://www.openstack.org. Accessed: 2016-08-26.

[9] Zabbix official website. URL http://www.zabbix.com. Accessed: 2016-08-26.

[10] Data center efficiency assessment. Technical report, Natural Resources Defense Council (NRDC), New York, NY, 2014.

[11] The foundation for better business intelligence. Technical report, Intel Corporation, 2014.

[12] How AWS pricing works. June 2015. URL https://d0.awsstatic.com/whitepapers/aws_pricing_overview.pdf.

[13] VMware vSphere 6 fault tolerance, architecture and performance. Technical report, VMware, January 2016. URL http://www.vmware.com/files/pdf/techpaper/VMware-vSphere6-FT-arch-perf.pdf.

[14] G. Aceto, A. Botta, W. De Donato, and A. Pescapè. Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115, 2013.

[15] G. Anders. Amazon's web-services delight. URL http://www.forbes.com/sites/georgeanders/2015/04/23/amazons-web-services-delight-16-9-margins-more-joy-ahead. Accessed: 2016-08-26.

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53 (4):50–58, Apr. 2010.

[17] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff. OpenPiton: An open source many-core research framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 217–232, New York, NY, USA, 2016. ACM.

[18] S. A. Baset. Cloud SLAs: Present and future. *SIGOPS Oper. Syst. Rev.*, 46(2):57–66, July 2012.

[19] R. Birke, L. Y. Chen, and E. Smirni. Data centers in the wild: A large performance study. 2012. URL http://domino.research.ibm.com/library/cyberdig.nsf/papers/0C306B31CF0D3861852579E40045F17F.

[20] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, and A. Y. Zomaya. Energy-efficient data replication in cloud computing datacenters. *Cluster Computing*, 18(1):385–402, 2015.

[21] K. D. Bowers, A. Juels, and A. Oprea. HAIL: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 187–198, New York, NY, USA, 2009. ACM.

[22] C. Cérin, C. Coti, P. Delort, F. Diaz, M. Gagnaire, Q. Gaumer, N. Guillaume, J. Lous, S. Lubiarz, J. Raffaelli, et al. Downtime statistics of current cloud solutions. *International Working Group on Cloud Computing Resiliency, Tech. Rep*, 2013.

[23] J. Chen, C. Wang, B. B. Zhou, L. Sun, Y. C. Lee, and A. Y. Zomaya. Tradeoffs between profit and customer satisfaction for service provisioning in the cloud. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 229–238, New York, NY, USA, 2011. ACM.

[24] X. Chen, C. D. Lu, and K. Pattabiraman. Failure analysis of jobs in compute clouds: A google cluster case study. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 167–177, Nov 2014.

[25] C. Colman-Meixner, C. Develder, M. Tornatore, and B. Mukherjee. A survey on resiliency techniques in cloud computing infrastructures and applications. *IEEE Communications Surveys Tutorials*, PP(99):1–1, 2016.

[26] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174. San Francisco, 2008.

[27] M. B. de Carvalho, R. P. Esteves, G. da Cunha Rodrigues, L. Z. Granville, and L. M. R. Tarouco. A cloud monitoring framework for self-configured monitoring slices based on multiple tools. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pages 180–184, Oct 2013.

[28] S. Fu. Failure-aware resource management for high-availability computing clusters with distributed virtual machines. *Journal of Parallel and Distributed Computing*, 70 (4):384–393, 2010.

[29] S. Fu and C. Z. Xu. Quantifying temporal and spatial correlation of failure events for proactive management. In *Reliable Distributed Systems, 2007. 26th IEEE International Symposium on*, pages 175–184, Oct 2007.

[30] S. Fu and C. Z. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, pages 1–12, Nov 2007.

[31] P. Garraghan, I. S. Moreno, P. Townend, and J. Xu. An analysis of failure-related energy waste in a large-scale cloud environment. *IEEE Transactions on Emerging Topics in Computing*, 2(2):166–180, June 2014.

[32] T. Gupta, J. B. Leners, M. K. Aguilera, and M. Walfish. Improving availability in distributed systems with failure informers. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 427–441, 2013.

[33] S. M. Habib, S. Ries, and M. Muhlhauser. Cloud computing landscape and research challenges regarding trust and reputation. In *Ubiquitous Intelligence Computing and 7th International Conference on Autonomic Trusted Computing*, pages 410–415, Oct 2010.

[34] P. Heidari, M. Hormati, M. Toeroe, Y. A. Ahmad, and F. Khendek. Integrating OpenSAF high availability solution with OpenStack. In *Services (SERVICES), 2015 IEEE World Congress on*, pages 229–236, June 2015.

[35] H. Hollimon. Survey: Software as a service perceptions survey, 2008. URL http://c1776742.cdn.cloudfiles.rackspacecloud.com/downloads/surveys/SaaSSurvey.pdf.

[36] A. Jahanbanifar, F. Khendek, and M. Toeroe. Providing hardware redundancy for highly available services in virtualized environments. In *Software Security and Reliability (SERE), 8th International Conference on*, pages 40–47, June 2014.

[37] M. Jammal, A. Kanso, and A. Shami. CHASE: Component high availability-aware scheduler in cloud computing environment. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 477–484, June 2015.

[38] B. Javadi, D. Kondo, J. M. Vincent, and D. P. Anderson. Discovering statistical models of availability in large distributed systems: An empirical study of SETI@home. *IEEE Transactions on Parallel and Distributed Systems*, 22(11):1896–1903, Nov 2011.

[39] B. Javadi, J. Abawajy, and R. Buyya. Failure-aware resource provisioning for hybrid cloud infrastructure. *Journal of Parallel and Distributed Computing*, 72(10):1318 – 1331, 2012.

[40] X. Ju, L. Soares, K. G. Shin, K. D. Ryu, and D. Da Silva. On fault resilience of OpenStack. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SoCC '13, pages 2:1–2:16, New York, NY, USA, 2013. ACM.

[41] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 398–407, May 2010.

[42] M. Krasich. How to estimate and use MTTF/MTBF would the real MTBF please stand up? In *Reliability and Maintainability Symposium, 2009.*, pages 353–359, Jan 2009.

[43] N. Limrungsi, J. Zhao, Y. Xiang, T. Lan, H. H. Huang, and S. Subramaniam. Providing reliability as an elastic service in cloud computing. In *Communications, IEEE International Conf. on*, pages 2912–2917, June 2012.

[44] O. Litvinski and A. Gherbi. OpenStack scheduler evaluation using design of experiment approach. In *16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*, pages 1–7, June 2013.

[45] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis. Virtualization for high-performance computing. *SIGOPS Oper. Syst. Rev.*, 40(2):8–11, Apr. 2006.

[46] S. Mittal, K. P. Joshi, C. Pearce, and A. Joshi. Automatic extraction of metrics from SLAs for cloud service management. In *IEEE Int. Conf. on Cloud Engineering*, 2016.

[47] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Key challenges in cloud computing: Enabling the future internet of services. *IEEE Internet Computing*, 17(4):18–25, July 2013.

[48] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive fault tolerance for hpc with Xen virtualization. In *Proceedings of the 21st Annual International Conference on Supercomputing*, ICS '07, pages 23–32, New York, NY, USA, 2007. ACM.

[49] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do internet services fail, and what can be done about it? In *USENIX symposium on internet technologies and systems*, volume 67. Seattle, WA, 2003.

[50] W. Pan, J. Rowe, and G. Barlaoura. Records in the cloud (RiC) user survey report, Oct 2013. URL https://open.library.ubc.ca/cIRcle/collections/42591/items/1.0075820.

[51] D. A. Patterson, G. Gibson, and R. H. Katz. *A case for redundant arrays of inexpensive disks (RAID)*, volume 17. ACM, 1988.

[52] C. Pham, P. Cao, Z. Kalbarczyk, and R. K. Iyer. Toward a high availability cloud: Techniques and challenges. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*, pages 1–6, June 2012.

[53] ProfitBricks Inc. Survey: What the cloud will mean for MSPs and the IT channel in 2016, 2015. URL http://info.profitbricks.com/rs/035-MXN-498/images/it-channel-survey-cloud-computing-2016-profitbricks.pdf.

[54] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., Mountain View, CA, USA, Technical Report*, 2011.

[55] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing elephants: novel erasure codes for big data. In *Proceedings of the 39th int. conf. on Very Large Data Bases*, PVLDB'13, pages 325–336. VLDB Endowment, 2013.

[56] D. J. Scales, M. Xu, M. D. Ginzton, et al. Low overhead fault tolerance through hybrid checkpointing and replay, July 30 2013. US Patent 8,499,297.

[57] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337–350, Oct 2010.

[58] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *FAST*, volume 7, pages 1–16, 2007.

[59] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM errors in the wild: A large-scale field study. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 193–204, New York, NY, USA, 2009. ACM.

[60] Y. Sharma, B. Javadi, and W. Si. On the reliability and energy efficiency in cloud computing. *Parallel and Distributed Computing 2015*, 27:111, 2015.

[61] M. Stevenson and M. Spring. Flexibility from a supply chain perspective: definition and review. *International Journal of Operations & Production Management*, 27(7):685–713, 2007.

[62] A. Türkyilmaz and C. Özkan. Development of a customer satisfaction index model: An application to the turkish mobile phone sector. *Industrial Management & Data Systems*, 107 (5):672–687, 2007.

[63] A. Undheim, A. Chilwan, and P. Heegaard. Differentiated availability in cloud computing SLAs. In *Grid Computing (GRID), 12th IEEE/ACM International Conference on*, pages 129–136, Sept 2011.

[64] G. Vallee, K. Charoenpornwattana, C. Engelmann, A. Tikotekar, C. Leangsuksun, T. Naughton, and S. L. Scott. A framework for proactive fault tolerance. In *Availability, Reliability and Security. ARES '08. Third International Conference on*, pages 659–664, March 2008.

[65] K. V. Vishwanath and N. Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 193–204, New York, NY, USA, 2010. ACM.

[66] Y. Wei and M. B. Blake. Service-oriented computing and cloud computing: Challenges and opportunities. *IEEE Internet Computing*, 14(6):72–75, 11 2010.

[67] Y. Xiaoyong, L. Ying, J. Tong, L. Tiancheng, and W. Zhonghai. An analysis on availability commitment and penalty in cloud sla. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, volume 2, pages 914–919, July 2015.

[68] X. Xu and H. H. Huang. DualVisor: Redundant hypervisor execution for achieving hardware error resilience in datacenters. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 485–494, May 2015.

[69] E. Yaqub, R. Yahyapour, P. Wieder, C. Kotsokalis, K. Lu, and A. I. Jehangiri. Optimal negotiation of service level agreements for cloud-based services through autonomous agents. In *Services Computing (SCC), IEEE International Conference on*, pages 59–66, June 2014.

[70] W. Zheng and X. Wang. Data center sprinting: Enabling computational sprinting at the data center level. In *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*, pages 175–184, June 2015.

[71] Y. Zhou and D. Wentzlaff. The sharing architecture: Sub-core configurability for iaas clouds. In *Proceedings of the 19th Int. Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 559–574, New York, NY, USA, 2014. ACM.

[72] Y. Zhou, H. Hoffmann, and D. Wentzlaff. CASH: Supporting iaas customers with a sub-core configurable architecture. In *Proceedings of the 43rd Annual International Symposium on Computer Architecture*, ISCA '16, New York, NY, USA, 2016. ACM.