An Approach to Deal with Processing Surges in Cloud Computing

Darlan Segalin, Altair Olivo Santin, João Eugenio Marynowski
Graduate Program in Computer Science
Pontifical Catholic University of Parana
Curitiba, Brazil
{darlan.segalin, santin, jeugenio}@ppgia.pucpr.br

Liandro Segalin
Computer Science Department
Federal University of South Border
Chapeco, Brazil
liandrose@hotmail.com

Carlos Maziero
Computer Science Department
Federal Technological University of Parana
Curitiba, Brazil
maziero@dainf.ct.utfpr.edu.br

***Abstract*—Processing surges are fast and unexpected changes in the processing demand that commonly occur in cloud computing. The cloud elasticity enables to handle processing surges, increasing and decreasing resources as required. However, a surge can be very fast, so that the overhead to provide more resource is greater than the processing benefit. On the other hand, if the surge is slow and continuous, and the required resources are not provided, the application performance may be impaired or interrupted. This paper presents a machine learning-based approach to detect and classify processing surges, in order to improve the cloud resource management, minimizing losses for the application and cloud provider. We use a real cloud dataset to select features, to construct the classifier and to test our approach, which successfully detected and classified 99% of the processing surges.**

Keywords—Processing Surge; Spike and Flash Crowds; Pattern Recognition; Support Vector Machine

I. INTRODUCTION

Cloud computing has been changing the way people use computers as well as the way services are provided. Cloud computing aims at using computational resources in any place, regardless the hardware infrastructure, using the Internet as a communication medium. Cloud computing dynamically provides resources following each client demand [1].

The resource dynamic provisioning typically involves: (i) to build a processing model for prospecting the required resources for each particular demand; (ii) periodically, to use the performance model to detect and to classify processing demands, and (iii) to automate the allocation of the required resources for each demand.

The processing model can be built using several techniques, including queueing theory [2], control theory [3] and statistical machine learning [4]. However, up to now remains an open issue detecting and classifying processing surges.

A processing surge can be classified as a spike when it waste less time to process a demand than to provide more resources.

A processing surge can be classified as a flash crowd when it demands an intense, continuous, and lasting usage of processing resource.

Since an effective resource demand is usually not known in advance, it becomes necessary to detect as early as possible whether a processing surge is a spike or a flash crowd. If the surge is a flash crowd and new resources are not allocated, the application performance may be impaired or even interrupted. On the other hand, if additional resources are allocated but the surge is a spike, resources and their costs are wasted.

A new resources allocation within the same physical server is often relatively cheap, and the dynamic resources allocation can be made even the surge is a spike. However, with a significant demand for dynamic allocations, the resource allocation must be made in another physical server or even in another cloud provider, and its viability can be only justified when the surge is a flash crowd.

The hypothesis of this work is that it is possible to detect and classify a processing surge as spike and flash crowd. We aim at appropriately serving each surge, without impairing the application processing or wasting resources. Moreover, the literature does not address this issue on cloud computing environment.

In this paper, we present an approach to detect and classify processing surges in cloud computing, in order to enhance the system overall performance and to optimize the dynamic allocation of resources. The approach is based on machine learning and uses SVM (Support Vector Machine) to analyze a processing workload for evaluating a surge as a spike or flash crowd.

Initially, during the detection phase and for non-impairing the application, we adopt a local strategy of dynamic allocation, assuming that local resources are enough to meet a spike demand. When the surge is identified as a flash crowd, resources are re-allocated to dynamically meet such processing demand.

The remainder of the paper was organized in the following way. Section II briefly reviews some aspects of machine learning. Section III discusses related work. Section IV presents our proposal and obtained results. Section V brings final considerations.

II. MACHINE LEARNING

A basic principle applied in machine learning is to learn from a set of examples (feature vectors that are labeled as a class). The learning based on this principle can be: (i) supervised – when an expert provides examples for each class, and (ii) unsupervised – when a technique identifies patterns or trends from an example set, clusters it and assigns them to a class [5].

A feature is a characteristic of the problem that differentiates a set of examples from other. The relevance of each feature under analysis is essential to apply machine learning. The selection of relevant features also can be made by an expert or through automated techniques for this purpose.

Figure 1 shows the vector labeling process, which consists of the input examples (dataset), the labeled vector (a set of features or attributes that are assigned to a class), and the classifier, a machine learning algorithm.

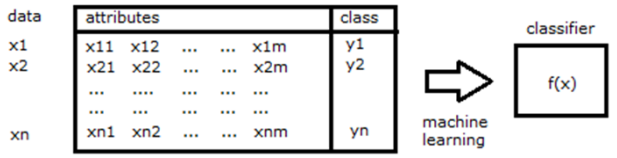

Figure 1. The vector labeling process for machine learning classification. Adapted from [6].

Labeled vectors are used in a training phase (employed to learn from input vectors) to build a model. After tested and validated, the model provides an accuracy, i.e., a success rate estimation for the model to correctly identify a class. The classification phase uses the model and a classifier algorithm to assign a class for input data.

The SVM (Support Vector Machine) classifier is an algorithm based on statistical learning [6]. SVM has been widely and successfully used in various areas of knowledge, including resource allocation in cloud computing for input data classification [7].

III. RELATED WORK

In the follow, we briefly consider the works of literature that proposes a way to circumvent allocation or relocation problem, in order to deal with the resource management difficulties.

Lagar et al. [8] address elasticity and resource management in cloud computing. They present an approach to clone a dozen of Virtual Machines (VMs) in less than one second, without specialized hardware and using a simple API programming interface. Bryan et al. [9] also address this issue, and they propose the cloud micro-elasticity. VM clones are instantaneous made based on "state coloring", enabling to analyze the VMs and identify similarity regions, and making easier the cloning process. Gulati et al. [10] address the load balancing issue and present some central challenges in developing cloud-computing scalability. The approaches [8, 9, 10] tray to reduce the negative impacts of allocation or relocation (copy of a VM in a new environment). However, they did not avoid the resource allocation if not need.

Eun-kyu et al. [11] offer a strategy to estimate the resource usage trough an algorithm to control workflows using the cloud elasticity. The algorithm determines the execution capacity of a workflow with a minimal resource and maximal capacity utilization to minimize costs of resource allocations and to meet deadlines. Bodik et al. [4] estimate the resource usage by a model and techniques of analysis based on Statistical Learning Machine (SLM). They aimed to solve problems of unrealistic performance models obtained by linear models and queueing theory. The proposals [4, 11] try to model or estimate resource demand, but they aim at performance instead of optimizing the resource allocation.

Liang et al. [13] report that most websites that become popular without expecting such popularity suffer from flash crowds. They use metrics to monitor the external usage of resources from such websites to thereby predict web traffic surges. Bodik et al. [12] use the changes in the network data volumes and objects individual popularities to predict surges. Both approaches [12, 13] characterize spike and flash crowds creating models and simulations for web services, but they consider and deal with only web traffic.

In summary, the literature efforts are made to decrease the impact of spikes and flash crowds in the system performance, also seeking to estimate the resource usage or trying to manage its impact. However, only web traffic is considered to deal with spike and flash crowd surges. None of the presented proposals analyze different features of processing and memory usage in an attempt to differentiate, in runtime, the processing surges (spikes and flash crowd).

IV. PROPOSAL

We propose a strategy to deal with processing surges in cloud computing and the use of an SVM classifier to identify spike and flash crowd surges. The dynamic resource allocation of our proposal follows the strategy of allocating new resources on local VMs, classifying a processing surge, and evaluating whether is necessary to migrate the VMs to a remote domain/infrastructure for attending the demand. Figure 2 depicts our proposal scheme.

Our allocation strategy is based on the fact of a small time is spent to instantiate a VM into the same physical machine (host), given a shared storage (resources for user data or system storage).

The process to identify a surge starts with collecting a features set, which is evaluated by a machine learning technique to classify the surge as a spike or flash crowd. If the surge is a spike, nothing is done because probably the surge already is over, and additional resources are no longer required. If the surge is a flash crowd, the VM is migrated to

another machine, which has resources to serve a more intense and lasting workload. This strategy was adopted due to the cost to migrate a VM to a remote host (via network), which may be within a private or public cloud (e.g. Amazon and Azure).

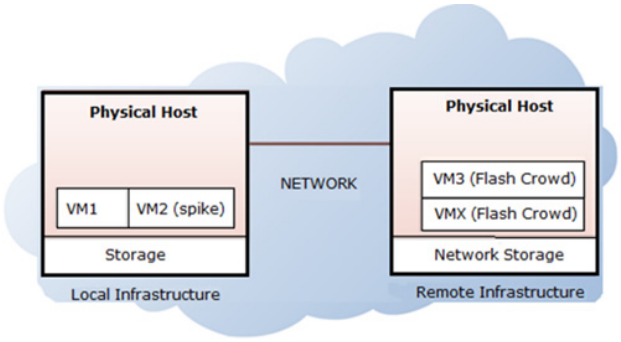

Figure 2. Overview of the proposal allocation scheme.

One of the main difficulties for differentiating processing surges is to characterize them as spike or flash crowd since both consume the entire CPU resources. Although their surge lengths are different, it is hard to find, at runtime, the threshold feature values that are able to distinguish a spike from a flash crowd. Section A presents a set of features used to analyze and to define a threshold. We collected features from a Brazilian data center that offers infrastructure as a service in cloud computing for customers around South America. The collected data represents average values observed in a single VM of the cloud environment, using the VMware hypervisor.

A. *Feature Selection*

We selected features from a workload produced in a controlled environment and using actual processing demands. We obtained the measured values considering the tested environment in various situations.

Initially, we considered thirty-eight features, but, after several manual analyzes, we removed some features because they remained constant along the time, hindering the classification. Table I shows the remaining twenty-three features and a brief description of each one.

TABLE I. MEMORY AND CPU FEATURES MANUALLY SELECTED.

Collected Metric Name	Description and Features Enumeration
CpuSystemTotalLoad	CPU System – Total Load (1)
CpuSystemLoad	CPU System – System Load (2)
CpuSystemLoadAvg	CPU System – Average Load (3)
CpuSystemUserLoad	CPU System – User Load (4)
CpuSystemWaiting	CPU System – Wait Load (5)
CpuPagesIn	CPU System – Pages Reads (6)
CpuInterruptTotal	CPU System – Total Interruptions (7)
CpuContextSwitch	CPU System – Switch of Context (8)
CpuCore01TotalLoad	CPU Core 01 – Total Load (9)
CpuCore01SystemLoad	CPU Core 01 – System Load (10)
CpuCore01UserLoad	CPU Core 01 – User Load (11)
CpuCore02TotalLoad	CPU Core 02 – Total Load (12)
CpuCore2SystemLoad	CPU Core 02 – System Load (13)
CpuCore02UserLoad	CPU Core 02 – User Load (14)
MemPhyUsed	Physical Memory Used (15)
MemPhyApplication	Physical Memory Used Application (16)
MemPhyCached	Physical Memory Used Cache (17)
MemPhyBuff	Physical Memory Used Buffer (18)
MemSwapUsed	Physical Memory Used Cache (19)
DiskChangeTotalAcc	Disk Access Total (20)
DiskRateTotalAcc	Disk Rate Total Access (21)
DiskRateReadAcc	Disk Rate Read Access (22)
DiskRateWriteAcc	Disk Rate Write Access (23)

We used System Monitor or System Guard [14] to collect 23 features of local VMs, storing them in text files. We collected the features records every one-second and for around 13.33 minutes – enough time to observe the occurrence of processing surges like spikes and flash crowds. We generated 800 records for each workload.

Figure 3 depicts a log fragment generated by the System Monitor for the System CPU total load feature (*CpuSystemTotalLoad*, Table I).

```
Mar 22 11:27:35 localhost cpu/system/TotalLoad: 100
Mar 22 11:27:36 localhost cpu/system/TotalLoad: 32.5
Mar 22 11:27:37 localhost cpu/system/TotalLoad: 21.0526
Mar 22 11:27:38 localhost cpu/system/TotalLoad: 28.2051
Mar 22 11:27:39 localhost cpu/system/TotalLoad: 23.0769
Mar 22 11:27:40 localhost cpu/system/TotalLoad: 28.2051
Mar 22 11:27:41 localhost cpu/system/TotalLoad: 22.5
```

Figure 3. Log fragment for the CpuSystemTotalLoad feature.

B. *Data Collection*

We obtained a normal workload by running LAMP (Linux, Apache, MySQL, and PHP) applications because they are considered typical in a local cloud VMs of a datacenter. Figure 4 presents the visualization of a typical workload obtained in this case.

One can noticed that the average CPU load is around 40% (ranging from 20% to 60%) and does not have a

particular pattern because the user behavior for processing demand is out of our control.

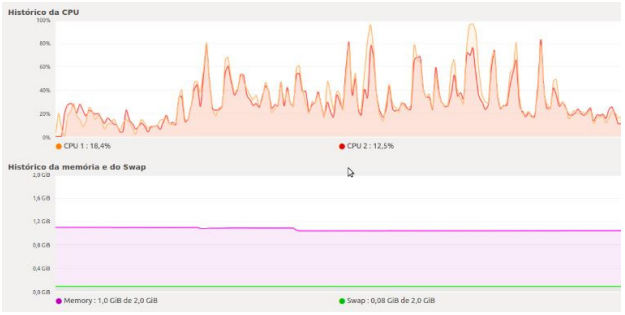

Figure 4. Normal workload.

We obtained a controlled workload by running a data encryption standard application with a 168 bits key encryption algorithm to consume memory and CPU. We used a shell script program to schedule the time intervals in which the application was executed. We developed 13 essays (Figure 5 to Figure 17).

Initially, the interval between generated processing surges (CPU processing below 20%) was 26 seconds, and the processing surge (CPU processing above 80%) was 2 seconds. In the next essays, we reduced the interval by 2 seconds and increased the surge processing time by the same 2 seconds.

This procedure enabled us to create processing surges from two seconds to twenty-six seconds. We stored the processing results in files named as "Spike" or "Flash" followed by a sequential number. Figure 5 to Figure 17 show the CPU and memory usage that were obtained from the System Monitor, in those essays.

In Figure 5, we start to control the CPU processing demand, assigning a workload with around 2 seconds of processing surge and around 26 seconds of the interval between generated processing surges.

Figure 5. Spike 01 workload.

In Figure 6, the intervals were around 24 seconds and the processing surges were around 4 seconds. In other essays (Figure 7 to Figure 17), we continue to increase the processing surge length and to reduce the intervals in 2 seconds.

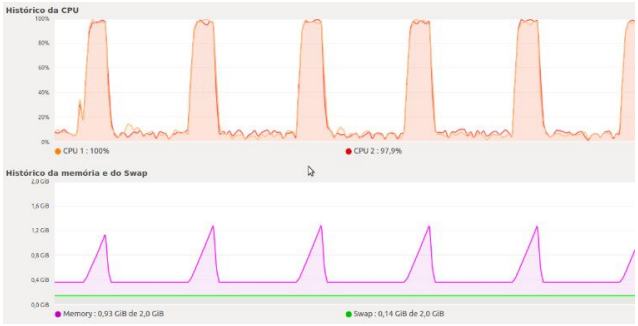

Figure 6. Spike 02 workload.

Figure 7. Spike 03 workload.

Figure 8. Spike 04 workload.

Figure 9. Spike 05 workload.

Figure 10. Spike 06 workload.

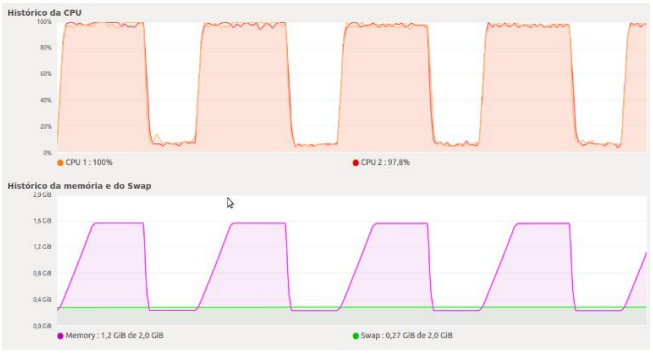

Figure 11. Spike 07 workload.

Figure 12 shows the last workload considered by a datacenter administration expert as a spike (Spike 08 workload). The intervals length has around 12 seconds, and the processing surge has around 16 seconds.

Figure 12. Spike 08 workload.

Figure 13 shows the first workload considered by datacenter expert as flash crowd. The surges considered as flash crowd start with around 10 seconds of intervals length and around 18 seconds of processing surges.

Figure 17 shows the last workload for flash crowd with around 2 seconds of intervals length and around 26 seconds of processing surges.

Figure 13. Flash 09 workload.

Figure 14. Flash 10 workload.

Figure 15. Flash 11 workload.

Figure 16. Flash 12 workload.

Figure 17. Flash 13 workload.

The information about the processing surge, given by a datacenter administration expert – to classify the surge as a spike or flash crowd, was used to label the feature vectors used in the SVM training phase.

The next section presents the process to get the surge model for the SVM classifier.

C. *Applying Machine Learning*

We developed a script to preprocess the System Monitor log file, to eliminate irrelevant information and to generate the feature vectors. Figure 18 shows an example of three lines (from 0 to 2) of a System Monitor log file.

Each line has the assigned numbers for each feature, from 1 to 23 (according to Table I), followed by a colon (':') and a feature value. For example, the highlighted '7: 1134.22' means that the feature 'CPU System - Total Interruptions (7)' has the value 1134.22. Figure 18 shows other examples regarding other features and processing values.

0 1: 18.1818 2: 10 3: 1.27 4: 25.9259 5: 0 6: 0 7: 2050.62 8: 2747.1 9: 0 10: 0 11: 0 12: 81.8182 13: 0 14: 30.7692 15: 2.03603e+06 16: 1.00913e+06 17: 873676 18: 152080 19: 0 20: 0 21: 0 22: 0 23: 0

1 1: 71.0526 2: 19.5122 3: 1.58 4: 76.9231 5: 0 6: 0 **7: 1134.22** 8: 38481.1 9: 100 10: 0 11: 66.6667 12: 0 13: 0 14: 92.3077 15: 1.07724e+06 16: 1.2555e+06 17: 445008 18: 84136 19: 0 20: 0 21: 0 22: 0 23: 0

2 1: 100 2: 25 3: 2.19 4: 76.25 5: 0 6: 0 7: 830.044 8: 19300.6 9: 100 10: 17.3913 11: 77.7778 12: 100 13: 0 14: 0 15: 2.55831e+06 16: 1.99384e+06 17: 476340 18: 88248 19: 0 20: 0 21: 0 22: 0 23: 0

Figure 18. File fragment with feature vectors.

After we had generated the feature vectors and their labeling (composing the dataset), we applied the SVM method to create a classifier.

The hypothesis was that the labeling made by an expert was correct. Moreover, the threshold of both surges (spike and flash) was successfully identified if the training and test set had obtained a success rate close to 100% for several surge combinations (spike and flash).

Otherwise, the labeling should be made again, meaning that the labeling once defined as spike could be a flash crowd or vice-versa, or even, that both surges could have been misunderstood with a normal demand. Therefore, the difficulty was to choose the surge workload combinations to test and to confirm the hypothesis, considering that it would be hard to test all the combinations of thirteen processing demands among themselves.

We used the *k*-fold cross-validation method to evaluate the generalization ability of the surge model to classify an independent dataset. The *k*-fold method divides the training dataset into *k* subsets. We assumed *k=3*, since we have three classes to train: normal, spike and flash crowd. Then, the training uses three vector subsets.

Figure 19 depicts the applied cross-validation scheme, using *3*-fold. The cross-validation process consists of repeating the tests three times. We used the average result to estimate the parameters and to calculate the model accuracy (success rate).

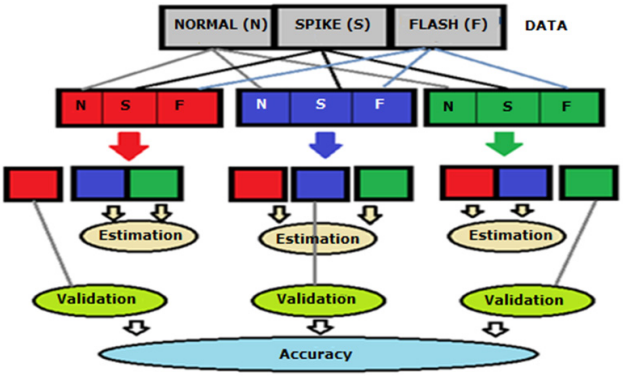

Figure 19. Applied cross-validation scheme.

All datasets were split in a proportion of 50% for the model training and 50% for the model testing, i.e., we used 400 records in the training phase and other 400 records in the testing phase.

Table II shows a set of tests that involves a combination of surge processing demands and the results obtained during the training and testing phases.

Initially, we considered the first six tests, from Test01 to Test06, and we concluded that spike and flash crowd were detectable and distinguishable, once the success rates were greater than 99%.

We used Test07 to Test12 to investigate the border between the spike and flash surges. Only Test12 presented the accuracy rate lower than 99%. Then, we executed Test13 and Test14, and Test14 got an error.

The Spike09 workload, that contained a previously labeled spike processing demand in Test13, was relabeled as Flash09 because its surge of processing demand was characterized as a flash crowd instead of a spike.

From Test01 to Test06 the processing demand was incrementally reduced. Test01 presented a surge characterized by extreme surges of processing demands, combining Spike01 with Flash13. After that, in each test a

lower surge of processing demands were considered, toward the border between spike and flash crowd surges.

TABLE II. TRAINING AND TESTING RESULTS FOR THE DATASET DEMANDS COMBINATION.

Test name	Demand	Cross-validation rate: training (%)	Accuracy rate: test (%)
Test01	Normal, Spike01, Flash13	100.00	100.00
Test02	Normal, Spike02, Flash12	99.92	99.92
Test03	Normal, Spike03, Flash11	99.34	99.83
Test04	Normal, Spike04, Flash10	99.92	100.00
Test05	Normal, Spike05, Flash09	100.00	100.00
Test06	Normal, Spike06, Spike08	100.00	99.33
Test07	Normal, Spike06, Spike07	100.00	100.00
Test08	Normal, Spike07, Spike08	100.00	100.00
Test09	Normal, Spike07, Flash09	100.00	100.00
Test10	Normal, Spike08, Flash09	100.00	100.00
Test11	Normal, Spike08, Flash10	100.00	99.92
Test12	Normal, Spike09, Flash10	100.00	88.67
Test13	Normal, Flash09, Flash11	100.00	99.92
Test14	Normal, Spike09, Flash12	99.75	91.83

The goal is to prove that the labeling made by the experts are confirmed when the model was obtained in training phase, and the success rate of the test vectors was kept high, i.e., greater than 99%.

The results in Table II show that, in all tests, the demands identified by the experts as spike, stored in files from Spike01 to Spike08, are confirmed as spikes, due to the success rates obtained.

Observe that the experts classified these vectors as spikes, and the training model and the tests confirmed the features vectors labeling. The model was generated from the examples extracted from respective collection files for the spike workload. We obtained the previously commented confirmation with the classifier tests and success rates greater than 99%.

In the subsequent workload after Spike08, there is a "gray region" mainly identified in the Flash09 workload. Based on the learning model provided by the classifier, the processing demand tends to indicate that a spike surge may be present in the workload that was predefined as a flash crowd by experts. This situation will be better discussed in Section D.

The proposal hypothesis would fail if the surge classification did not occur with high accuracy, so a spike would not be easily distinguishable from a flash crowd as the literature suggests [12, 13].

In other words, labeling would be incorrect, or the features vectors would not represent features for characterizing spike and flash crowd surges. This issue will be further explored in Section E.

D. Analyzing the Datasets Vectors

Based on the results reported in Section C, we mixed all feature vectors (files) and created a single dataset with 1/3 of the records from each workloads (normal, spike, and flash crowd).

We executed the training with 50% of the new dataset, running the *k*-fold cross-validation with *k=3*, and the others 50% of the new dataset we reserved for the model testing. Table III shows the obtained results for the cross-validation and accuracy rates, considering the new dataset, i.e., all processing demand surges together.

TABLE III. RESULTS CONSIDERING ALL PROCESSING SURGES TOGETHER.

Test no.	Demands	Cross-Validation rate: training (%)	Accuracy rate: test (%)
Test00	Normal, Spike01-spike08, Flash09-Flash13	98.82	98.41 (5511/5600)

The goal of this evaluation was to ensure that all collected feature vectors correctly represent spike, flash crowd, and normal processing demands. All previously reported tests had considered these processing demands and they were separately recorded in the Spike01 to Flash13 files. Then, if the result of this evaluation does not generate a high success rate, many noises could be collected.

A noise would be wrong labeled records, which did not represent the processing demand in a discriminant way. The test results showed the dataset validity to represent the problem under study. They present a low noise rate (2.18%) in the total of collected and labeled vectors for all processing workloads (1 normal, 8 spikes, and 5 flash crowds).

E. Feature Selection Filter

We executed feature selection to reduce the noise (the classification imprecision) and to improve the classifier performance. We used the 23 features presented in Section A to apply the feature selection filter provided by Weka tool (www.cs.waikato.ac.nz/ml/weka/).

The Weka filter uses data characteristics to evaluate and select the features. Appling the Weka filter allows us to reduce from 23 to 15 the number of features. Table IV shows the discriminant features obtained after using the Weka filter.

We repeated all tests from Table II using the 15 selected features. The results (Table V) show a little improvement from Test01 to Test11 since the success rate were already excellent. However, in this case, the tests were made again to ensure that the 15 features did not negatively affect the previous results (considering the 23 features, Table I).

TABLE IV. SELECTED FEATURES OBTAINED BY WEKA FILTER.

Collected Metric Name	Description
CpuSystemTotalLoad	CPU System – Total Load (1)
CpuSystemLoad	CPU System – System Load (2)
CpuSystemLoadAvg	CPU System – Average Load (3)
CpuSystemUserLoad	CPU System – User Load (4)
CpuSystemWaiting	CPU System – Wait Load (5)
CpuCore01TotalLoad	CPU Core 01 – Total Load (9)
CpuCore01SystemLoad	CPU Core 01 – System Load (10)
CpuCore01UserLoad	CPU Core 01 – User Load (11)
CpuCore02TotalLoad	CPU Core 02 – Total Load (12)
CpuCore2SystemLoad	CPU Core 02 – System Load (13)
CpuCore02UserLoad	CPU Core 02 – User Load (14)
MemPhyUsed	Physical Memory Used (15)
MemPhyApplication	Physical Memory Used Application (16)
MemPhyCached	Physical Memory Used Cache (17)
MemPhyBuff	Physical Memory Used Buffer (18)

The main goal was to precisely study the behavior of Flash09 workload since the imprecision resulting from the classification in Test12 and Test14 from Table II. Once, the imprecise classification caused doubts about the border between spike and flash crowd surges, defined by an expert.

The selection of features was used to reduce the doubt related to the noise resulting from features that could be hindering the classification since they were not discriminant enough.

As we did not identify a problem related to the features, since the classification results were improved, we re-executed Test12 and Test14. Then, we considered the examples labeled as flash crowd by the experts, and that were relabeled as spike. The goal was to evaluate whether the results were improved, in order to verify a mistaken feature vector labeling hypothesis.

The results remained approximately the same, which enabled us to conclude that the problem was not with irrelevant features, neither about wrong labeling. Therefore, the hypothesis failure reported in Section C was not confirmed.

Thus, we concluded that the Flash09 workload actually represents a border between spike and flash crowd, due to the insignificant changes in the results of the success rate (Table II and Table V).

Test15 represents the same test Test00 from Table III, but considering the 15 selected features. In Test16, we took out the vectors of the Flash09 workload from the dataset of training and test, and we observed an increased classification with success rate, what clearly shows that the records of Flash09 workload mix spike and flash crowd vectors. Then, it has been proven that Flash09 workload contains vectors that identify the border between both processing demands. Therefore, the hypothesis of this study has been confirmed: spike and flash crowd are detectable and distinguishable.

TABLE V. TRAINING AND TESTING RESULTS FROM DEMANDS COMBINATION AND REDUCED FEATURES

Test no.	Demands	*Cross-Validation rate*: training (%)	*Accuracy rate*: test (%)
Test01	Normal, Spike01, Flash13	100.00	100.00
Test02	Normal, Spike02, Flash12	100.00	100.00
Test03	Normal, Spike03, Flash11	99.91	99.91
Test04	Normal, Spike04, Flash10	100.00	100.00
Test05	Normal, Spike05, Flash09	100.00	100.00
Test06	Normal, Spike06, Spike08	100.00	100.00
Test07	Normal, Spike06, Spike07	100.00	100.00
Test08	Normal, Spike07, Spike08	100.00	100.00
Test09	Normal, Spike07, Flash09	100.00	100.00
Test10	Normal, Spike08, Flash09	99.83	99.83
Test11	Normal, Spike08, Flash10	100.00	100.00
Test12	Normal, Spike09 (label changed), Flash10	92.5	91.41
Test13	Normal, Flash09, Flash11	100.00	97.83
Test14	Normal, Spike09 (changed label), Flash12	91.66	92.66
Test15	Normal, Spike01-Spike08, Flash09, Flash10-Flash13	98.53	98.58
Test16	Normal, Spike01-Spike08, Flash10-Flash13	99.34	99.34

The best results for the classification accuracy were obtained after the features selection – that eliminated eight features considered in the evaluation made through manual features selection, which improved the results obtained previously.

An important result from this tests phase was the confirmation that the features selection and the processing demands labeling were precise.

After the development of this work, it was concluded that only fifteen features, which refer to the CPU usage and memory, are necessary to differentiate spike (from Spike01 to Spike08 workloads) from flash crowd (from Flash10 to Flash13 workloads).

The processing demand collected in Flash09 is not conclusive and, therefore, should not be considered in the characterization of any processing surge. Thus, we consider the values of the Flash09 workload as the border between both kinds of processing surges.

In summary, if a surge is a spike, a local VM is instantiated. If the surge is a flash crowd, the VM can be migrated to another host, maybe in a public cloud infrastructure. This strategy minimizes the costs for the cloud

provider without impairing the application performance since the workload of VM provisioning that is unnecessary will be minimized or eliminated.

V. Conclusion

This paper presented an approach to deal with processing surges in cloud computing. We used machine learning to detect and classify spike and flash crowd surges in processing demands, an issue the literature has not addressed.

Data center experts have pre-characterized spikes and flash crowd surges, and we tuned such characterization through a surge model obtained by applying an SVM algorithm. The model enabled us to find a border to distinguish processing surges.

We applied Weka feature selection filter to reduce and improve the classifier performance and the classification rate. The filter eliminated eight features that were taken into account in the manual selection, improving the previous results significantly.

We concluded that only fifteen features, which refer to CPU and memory usage, are required to differentiate spike from flash crowd surges. The tests phase enabled us to confirm that the features and the labeling of demands were precise, exempting the Flash09 workload, which is not conclusive. Thus, a mixed vector of spike and flash crowd in a particular demand enabled us to define the values of those features as the border between both demands of processing surges.

We also introduced a strategy to improve cloud computing elasticity performance, minimizing the costs for the cloud provider without impairing the application.

The strategy is to allocate additional resource locally and after remotely, depending on the detected processing surge. If a surge is a spike, it is processed in a local datacenter VM. If the surge is a flash crowd, the VM can be migrated to another host (maybe in a public cloud infrastructure). Therefore, the VM migration for an unnecessary processing surge (spike) will be minimized or eliminated.

In future studies, we are looking towards to confirm that the use of this approach in load balancing can reduce the overhead in other service providers.

Acknowledgement

This work was partially sponsored by the Brazilian National Council for Scientific and Technological Development (CNPq), grants 310671/2012-4 and 404963/2013-7. Darlan Segalin thanks Eduardo Viegas for helping with Machine Learning and the Coordination for the Improvement of Higher Level Personnel (CAPES) for the scholarship granting.

References

[1] Buyya, R., Chee Shin Yeo, Venugopal, S. Market, "Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities", in Proc. of the HPCC, 2008.

[2] Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., and Wood, T. "Agile dynamic provisioning of multi-tier Internet applications", ACM Trans. Auton. Adapt. Syst., vol. 3, no. 1, 2008.

[3] Kalyvianaki, E., Charalambous, T. and Hand, S., "Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters", in Proc. of the ICAC, 2009.

[4] Bodík, P., Griffith, R., Sutton, C., Fox, A., Jordan, M. and Patterson, D., "Statistical machine learning makes automatic control practical for Internet datacenters", in Proc. of the USENIX HotCloud, 2009.

[5] Bishop, C. M. "Pattern recognition and machine learning", New York: Springer, vol. 1, 2006.

[6] Vapnik, V. "The nature of statistical learning theory", 2nd ed., Springer Verlag, ISBN: 0387987800, 1999.

[7] Krieger, A. and Simon, J., "Autonomic Resource Management with Support Vector Machines", in Proc. of the Lyon Conference, 2011.

[8] Lagar-Cavilla, H. A., Whitney, J., Scannel, A., Patchin, P., Rumble, S. M., Lara, E., Brudno, M. and Satyanarayanan, M., "SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing", in Proc. of the EuroSys, 2009.

[9] Bryant, R., Tumanov, A., Irzak, O., Scannell, A., Kaustubh, J., Hiltunen, M., Lagar-Cavilla, A., and Lara, E., "Kaleidoscope: Cloud Micro-Elasticity via VM State Coloring", AT&T Labs Research, University of Toronto, 2011.

[10] Gulatti, A., Shanmuganathan, G., Ahmad, I. and Holler, A., "Cloud Scale Resource Management: Challenges and Techniques", Vmware Labs, 2009.

[11] Eun-kyu, B., Yang-Suk, K., Jin-Soo, K. and Seungryoul, M., "Cost optimized provisioning of elastic resources for application workflows", Korea Advanced Institute of Science and Technology, Oracle, 2010.

[12] Bodik, P., Fox, A., Franklin, M. J., Jordan, M. I. and Patterson, D. A., "Characterizing, Modeling, and Generating Workload Spikes for Stateful Services", EECS Department, UC Berkeley, Berkeley, CA, 2010.

[13] Liang, C., Hiremagalore, S., Stavrou, A. and Rangwala, H., "Predicting Network Response Times Using Social Information", Center for Secure Information Systems, George Mason University,Technical Reports, 2007.

[14] "KDE System Monitor – ksysguard". Available: http://userbase.kde.org/KSysGuard