07-01-2020

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΩΝ

2ο ΠΑΡΑΔΟΤΕΟ

ΔΗΜΗΤΡΑΚΟΠΟΥΛΟΣ ΝΙΚΟΛΑΟΣ 21821

ΒΛΑΣΟΠΟΥΛΟΣ ΜΙΧΑΗΛ 21810

ΑΠΟΣΤΟΛΙΔΗΣ ΘΑΝΟΣ 21807

CONTENTS

ΕΡΩΤΗΜΑ 1

VIRTUALIZATION

Το virtualization ή αλλιώς η εικονικοποίηση, είναι η τεχνολογία που δημιουργεί ένα στρώμα αφαιρετικότητας πάνω από φυσικούς πόρους υπολογιστών (hardware components) και εξομοιώνει τις λειτουργίες τους. Αποτελεί την κυρίαρχη τεχνολογία ανάμεσα σε πολλές που χρησιμοποιεί το Cloud Computing με κύριο σκοπό τον διαμοιρασμό και την αξιοποίηση ακριβού και ισχυρού υλικού μεταξύ πολλών εικονικών μηχανών. Προσφέρει επίσης μεγάλη αξιοπιστία στην απόδοση αφού οι εφαρμογές μπορούν δυναμικά να αυξήσουν τους απαιτούμενους πόρους τους.

Για την δημιουργία εικονικών μηχανών αλλά και την ορθή λειτουργία τους χρησιμοποιούνται ειδικά λογισμικά που ονομάζονται hypervisors και παίρνουν τον ρόλο των “επόπτων”. Βρίσκονται σε ένα στρώμα “πάνω” από το λειτουργικό σύστημα των υπολογιστών, δημιουργούν και διαχειρίζονται εικονικές μηχανές και διαχωρίζουν τους φυσικούς πόρους από τα εικονικά περιβάλλοντα.

Πηγές:

[1] https://www.redhat.com/en/topics/virtualization/what-is-virtualization

[2] https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor

FAULT TOLERANCE

Με τη τεράστια εξάπλωση του cloud computing καθώς και τη ραγδαία αύξηση της χρήσης του σε ποικίλες περιπτώσεις, η ανάγκη διατήρησης του QoS (Quality of Service), της σταθερότητας, της αποτελεσματικότητας και την ασφάλειας, είναι συνεχώς αυξανόμενη. Σε μια τέτοια πραγματικότητα, η εμφάνιση των ανεπιθύμητων λαθών και συμπεριφορών αυξάνεται αντίστοιχα. Η δυνατότητα ενός συστήματος να αντιμετωπίσει αποτελεσματικά τα προβλήματα αυτά ονομάζεται Fault Tolerance.

Γενικότερα, ονομάζουμε μια προσέγγιση ως προσέγγιση «γενικού σκοπού», όταν αυτή στοχεύει να αντιμετωπίσει τα προβλήματα ενός συγκεκριμένου stack της εφαρμογής, αποκρύπτοντας πλήρως τα προβλήματα αυτά αλλά και τον μηχανισμό επίλυσης, από τον τελικό χρήστη. Εδώ, έχουν σημασία δύο παράμετροι:

• Τα σφάλματα προκύπτουν σε άγνωστο και τυχαίο χρόνο

• Το σύστημα έχει σχεδιαστεί ανεξάρτητο, μη λαμβάνοντας υπόψη τα ρίσκα αποτυχίας και πιθανών σφαλμάτων.

Μια τεχνική γενικού σκοπού, αποτελεί το Process Checkpointing. Ο στόχος της τεχνικής αυτής αποτελεί η σωστή, ασφαλής και σταθερή αποτύπωση ενός στιγμιότυπου της κατάστασης του συστήματος, τη δεδομένη χρονική στιγμή. Στη πραγματικότητα, μια εφαρμογή αποτελείται από πολλά threads σε

επίπεδο χρήστη αλλά και σε επίπεδο συστήματος, δηλαδή λειτουργεί σε παράλληλα επίπεδα. Η συνήθης υλοποίηση του process checkpointing περιέχει έναν μηχανισμό προσωρινού κλειδώματος, παγώνοντας έτσι στιγμιαία όλα τα εκτελούμενα threads, ώστε να αποτυπώσει και να αποθηκεύσει τη συγκεκριμένη κατάσταση του συστήματος. Σε ένα μελλοντικό επερχόμενο σφάλμα, η εκάστοτε εφαρμογή μπορεί να επιστρέψει στη προηγούμενη ασφαλή κατάσταση της και να συνεχίσει κανονικά την εκτέλεση της. Όλα αυτά γίνονται σε πολύ μικρό χρονικό διάστημα, μη αισθητό από τον χρήστη, δίνοντας έτσι την εντύπωση πως το σφάλμα δεν συνέβη ποτέ. Σε περίπτωση που γίνει κάτι τέτοιο αντιληπτό, τότε αφορά κάποιο σοβαρότερο σφάλμα, όμως η ασφάλεια τον δεδομένων καθώς και η ταχύτατη επιστροφή στη κανονική λειτουργία είναι εγγυημένη. Τέλος, μια process checkpointing τεχνική διακρίνεται με βάση 3 παραμέτρους:

• Σε ποιο stack της εφαρμογής υλοποιείται

• Με ποιο τρόπο υλοποιείται στο συγκεκριμένο stack

• Με ποιο τρόπο πραγματοποιείται η αποθήκευση του checkpoint

[Book] : Fault Tolerance Techniques for High-Performance Computing

By Thomas Herault & Yves Robert / Springer

ELASTICITY

ΟΡΙΣΜΟΣ

Η **ελαστικότητα (elasticity)** ορίζεται ως η ικανότητα που έχει ένα σύστημα στο να μπορεί να προσθέτει ή να αφαιρεί δυναμικά πόρους (όπως για παράδειγμα CPU πυρήνες, μνήμη, VM και container instances), ώστε να μπορεί να προσαρμόζεται στις διακυμάνσεις του φόρτου που υπάρχουν σε πραγματικό χρόνο. Η ελαστικότητα συχνά συσχετίζεται με όρους όπως η **επεκτασιμότητα (scalability)** και η **αποδοτικότητα (efficiency)** αλλά η ερμηνεία αυτών των όρων διαφέρουν μεταξύ τους. Η διαφορά μεταξύ ελαστικότητας και επεκτασιμότητας είναι ότι η δεύτερη δεν παίρνει ως παράμετρο τον χρόνο. Μπορεί να θεωρηθεί ότι το elasticity είναι η αυτοποίηση του scalability και ότι «χτίζεται» πάνω σε αυτό. Η **αποδοτικότητα**, από την άλλη, είναι ένα μέτρο του πόσο αποδοτικά χρησιμοποιεί μια εφαρμογή τους πόρους που διαθέτει. Σε ένα ιδανικό σενάριο, η εφαρμογή πρέπει να δεσμεύει ακριβώς τους πόρους που χρειάζονται για να ολοκληρωθεί μία διεργασία στον ζητούμενο χρόνο. Από τον ορισμό αυτόν, συνεπάγεται ότι η όσο αυξάνεται η ελαστικότητα ενός συστήματος, τόσο αυξάνεται και η αποδοτικότητα του.

ΤΕΧΝΙΚΕΣ ΕΛΑΣΤΙΚΟΤΗΤΑΣ

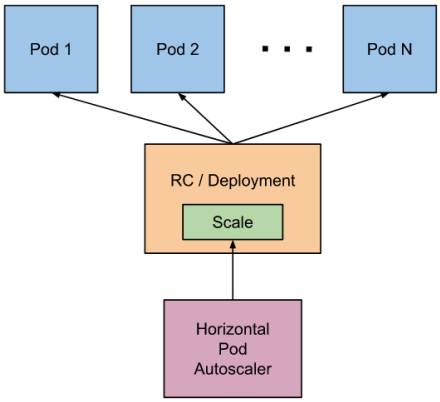
Υπάρχουν 2 κύριες τεχνικές με τις οποίες μπορεί να εφαρμοστεί η ελαστικότητα: **μέσω vertical ή horizontal scaling**.

- **Horizontal Scaling:** Η τεχνική με την οποία προστίθεται ή αφαιρούνται instances που περιέχουν ακριβώς την ίδια εφαρμογή, ανάλογα με το συνολικό φόρτο που υπάρχει μια δεδομένη χρονική στιγμή. Το traffic διανέμεται σε αυτά τα instances με τη χρήση Load Balancers.

- **Vertical Scaling:** Η τεχνική με την οποία προστίθεται ή αφαιρούνται πόροι από ένα συγκεκριμένο instance κατά το runtime.

Ο συνδυασμός αυτών των 2 τεχνικών αποτελεί επίσης επιλογή.

Η ΕΛΑΣΤΙΚΟΤΗΤΑ ΣΤΗΝ ΣΥΓΧΡΟΝΗ ΕΠΟΧΗ (KUBERNETES HORIZONTAL POD AUTOSCALER)



Ένα παράδειγμα horizontal scaling το οποίο αντιδρά σε σχέση με τον φόρτο είναι ο Horizontal Pod Autoscaler (HPA) του Kubernetes. Κάθε 15 δεύτερα (εκτός κι αν γίνει αλλαγή στην default τιμή), ένας controller ελέγχει τις μετρικές κάθε pod (CPU Utilization στην πιο απλή περίπτωση) και αν βρεθεί ότι παραβιάζονται κάποιες ή όλες από αυτές, τότε αυτόματα θα γίνει ένα scaling και θα δημιουργηθούν νέα pods. Μετά από κάποια περίοδο χρόνου, αν και όταν τα έξτρα pods είναι πλέον περιττά, ο HPA αυτόματα και ομαλά θα κάνει scaledown.

Μία πολύ βασική εκδοχή της φόρμουλας που αποφασίζει πόσα pod instances είναι ιδανικό να υπάρχουν είναι η εξής:

desiredReplicas = ceil[currentReplicas * (currentMetricValue / desiredMetricValue)]

Ετσί λοιπόν, αν για παράδειγμα το CPU Utilization είναι 2 φορές πάνω από το επιθυμητό, τα pods θα διπλασιαστούν για να εξισσοροπήσουν τη μετρική στο επιθυμητό επίπεδο. Είναι σημαντικό να σημειωθεί ότι υπάρχει και μια μεταβλητή (με default τιμή το 0.1) που ορίζει το tolerance του αλγορίθμου. Έτσι λοιπόν αν η απόκλιση των μετρικών από το ιδανικό απέχουν μόνο τόσο όσο ορίζει το tolerance, το HPA δεν θα ενεργοποιηθεί.

ΠΗΓΕΣ:

[1] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah and P. Merle, "Elasticity in Cloud Computing: State of the Art and Research Challenges," in *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 430-447, 1 March-April 2018, doi: 10.1109/TSC.2017.2711009.

[2] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

ΕΡΩΤΗΜΑ 2

Για αυτό το ερώτημα υλοποιήθηκε:

- Ένας μικρός SLA Parser βασιζόμενος σε κώδικα του CloudSim Plus (SlaContract.java)
- Μια μικρή Utility κλάση η οποία υπολογίζει τα violations που υπάρχουν(.

Παράδειγμα εξόδου (task_2_output.txt):

Simulation finished in: 106,43 seconds.

Total tasks executed: 555

Average MI: 37824,90

Average execution latency: 41,64

MI 90th percentile: 93899,60

Average tail execution latency: 93,55

Maximum task execution time violations: 187

ΕΡΩΤΗΜΑ 3

- Βλέποντας το παράδειγμα που υπάρχει στο **HostFaultInjectionExample1,** καταφέραμε να ενσωματώσουμε μία προσομοίωση faults σε νέα κλάση CustomFaultSimulation. Δημιουργήθηκαν επίσης και τα αντίστοιχα τεστ που επιβεβαιώνουν το αποτέλεσμα.

SLA OK:

Παράδειγμα εξόδου (task_3_sla_ok.txt)

Simulation finished in: 71,93 seconds.

Total tasks executed: 504

Average MI: 41161,79

Average execution latency: 21,90

MI 90th percentile: 94790,50

Average tail execution latency: 50,41

Maximum task execution time violations: 59

Mean Number of Failures per Hour: 100,000 (1 failure expected at each 0,01 hours).

Number of Host faults: 3

Number of VM faults (VMs destroyed): 0

Time the simulations finished: 0,0200 hours

Mean Time To Repair Failures of VMs in minutes (MTTR): 0,00 minute

Mean Time Between Failures (MTBF) affecting all VMs in minutes: 0,00 minutes

Hosts MTBF: 0,00 minutes

Availability: 100,00%

Max faults allowed by SLA 5,00

Faults: 3,00. SLA contract ok

FaultCustomSimulation finished!

SLA VIOLATED:

Παράδειγμα εξόδου (task_3_sla_violated.txt)

Simulation finished in: 230,15 seconds.

Total tasks executed: 463

Average MI: 37467,78

Average execution latency: 23,33

MI 90th percentile: 64466,40

Average tail execution latency: 82,63

Maximum task execution time violations: 64

Mean Number of Failures per Hour: 100,000 (1 failure expected at each 0,01 hours).

Number of Host faults: 10

Number of VM faults (VMs destroyed): 0

Time the simulations finished: 0,0639 hours

Mean Time To Repair Failures of VMs in minutes (MTTR): 0,00 minute

Mean Time Between Failures (MTBF) affecting all VMs in minutes: 0,00 minutes

Hosts MTBF: 0,00 minutes

Availability: 100,00%

Max faults allowed by SLA 5,00

Faults: 10,00. Out of SLA

FaultCustomSimulation finished!

ΕΡΩΤΗΜΑ 4 (BONUS)

- Τροποποιήθηκε η κλάση CustomSimulation ώστε να δημιουργεί VMs στα οποία εφαρμόζεται horizontal scaling αν ένα predicate ισχύει:

```
 private List<Vm> createListOfScalableVms(final int numberOfVms) {
    List<Vm> newList = new ArrayList<>(numberOfVms);
    for (int i = 0; i < numberOfVms; i++) {
        Vm vm = createVm();
        createHorizontalVmScaling(vm);
        newList.add(vm);
    }

    return newList;
}


private void createHorizontalVmScaling(Vm vm) {
    HorizontalVmScaling horizontalScaling = new HorizontalVmScalingSimple();
    horizontalScaling
            .setVmSupplier(this::createVm)
            .setOverloadPredicate(this::isVmOverloaded);
    vm.setHorizontalScaling(horizontalScaling);
}

private boolean isVmOverloaded(Vm vm) {
    return vm.getCpuPercentUtilization() > 0.6;
}
```

Παράδειγμα εξόδου (task_4_output.txt) :

INFO 0,51: HorizontalVmScalingSimpleVm 0: Requesting creation of Vm 6 to receive new Cloudlets in order to balance load of Vm 0. 0 CPU usage is 100.0%

INFO 0,51: DatacenterBrokerLoadBalancer1: List of 1 VMs submitted to the broker during simulation execution. VMs creation request sent to Datacenter.

INFO 0,51: HorizontalVmScalingSimpleVm 1: Requesting creation of Vm 7 to receive new Cloudlets in order to balance load of Vm 1. 1 CPU usage is 100.0%

INFO 0,51: DatacenterBrokerLoadBalancer1: List of 1 VMs submitted to the broker during simulation execution. VMs creation request sent to Datacenter.

INFO 0,51: HorizontalVmScalingSimpleVm 2: Requesting creation of Vm 8 to receive new Cloudlets in order to balance load of Vm 2. 2 CPU usage is 100.0%

INFO 0,51: DatacenterBrokerLoadBalancer1: List of 1 VMs submitted to the broker during simulation execution. VMs creation request sent to Datacenter.

INFO 0,51: HorizontalVmScalingSimpleVm 3: Requesting creation of Vm 9 to receive new Cloudlets in order to balance load of Vm 3. 3 CPU usage is 100.0%

INFO 0,51: DatacenterBrokerLoadBalancer1: List of 1 VMs submitted to the broker during simulation execution. VMs creation request sent to Datacenter.