20 Μαρτίου 2022

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Fogify Topology Generator

Εξαμηνιαία εργασία

Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων

Ιάσονας Ζαράνης
ΑΜ: 03115729
el15729@central.ntua.gr

Στέφανος Πατμανίδης
ΑΜ: 03109205
el09205@central.ntua.gr

Επιβλέπων:

Μωυσής Συμεωνίδης
Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου
symeonidis.moysis@cs.ucy.ac.cy

Ο πηγαίος κώδικας για την εργασία μας είναι διαθέσιμος στο GitHub: https://github.com/StefanPatman/asps22

I. ΕΙΣΑΓΩΓΗ

Το στήσιμο πραγματικών τοπολογιών είναι μια διαδικασία κοστοβόρα και χρονοβόρα, ειδικά αν χρειαστεί να γίνουν αλλαγές και βελτιώσεις σε πραγματικά συστήματα. Η χρήση ενός λογισμικού προσομοίωσης τοπολογιών μπορεί να λύσει αυτό το πρόβλημα και να διευκολύνει σημαντικά την σχεδίαση του συστήματος. Επιτρέποντας την δημιουργία σεναρίων προσομοίωσης με βάση παραμετροποιήσιμων τοπολογιών, μπορει να γινει διεξοδική δοκιμή και συγκριση των χαρακτηριστικων του συστηματος (όπως latency-delay, bandwidth κλπ) για κάθε σενάριο και να βρεθεί κατάλληλη ρύθμιση για το υπό εξέταση σύστημά.

Ένα εργαλείο ανοιχτού κώδικα που βοηθά με το παραπάνω είναι το Fogify [1]. Το Fogify προσφέρει ένα επεκτάσιμο πλαισιο για μοντελοποιηση και γρηγορο πειραματισμό υπηρεσιών ομίχλης μεσω επεκτάσιμης προσομοίωσης. Η προσομοίωση κάθε σεναρίου απαιτεί χειροκίνητη δημιουργια τοπολογιας για κάθε ένα από αυτά. Αυτή η διαδικασία μπορει να ειναι επιπονη και χρονοβόρα,

ειδικά σε περιπτώσεις επαναλαμβανόμενων δοκιμών (repeatable testing).

Σκοπός της εργασίας είναι η υλοποίηση μιας γεννήτριας τοπολογιών για το Fogify, χρησιμοποιώντας το Ether [2] ένα εργαλείο Python για δημιουργία των τοπολογιών. Για την παρουσίαση της υλοποίησης μας και την διεκπεραίωση πειραμάτων κατασκευάσαμε μια multiservice Edge εφαρμογή μετρήσεων θερμοκρασίας σε εργοστασιακό περιβάλλον. Χρησιμοποιήσαμε την γεννήτρια τοπολογιών και να ρυθμίσουμε και να δοκιμάσουμε διαφορετικά σενάρια προσομοίωσης.

II. ΕΦΑΡΜΟΓΗ

A. Περιγραφή της εφαρμογής

Το σύστημά μας αποτελείται από τους υπολογιστικούς κόμβους generators (παραγωγείς), aggregators (συλλέκτες) και processors(servers). Οι generators αποτελούν το πρώτο επίπεδο του συστήματος, είναι οι παραγωγοί των δεδομένων και συνήθως είναι iot devices ή services. Στο ακριβώς επόμενο επίπεδο βρίσκονται οι aggregators, οι οποίοι αποτελούν το edge επίπεδο του συστήματος. Συλλέγουν τα δεδομένα από τους generators, τα επεξεργάζονται σε μικρό βαθμό βγάζοντας περιοδικά τον μέσο όρο ανά generator των τιμών που τους στέλνονται και τον προωθούν στους processors. Ουσιαστικά κάνουν μια προεπεξεργασία των δεδομένων συγκρίνοντάς τα. Οι processors αποτελούν το τελευταίο και ιεραρχικά ψηλότερο επίπεδο, γνωστό και ως cloud. Εκεί καταλήγουν όλα τα δεδομένα συμπυκνωμένα πια από τους aggregators. Εδώ τα δεδομένα είναι ελεύθερα για χρήση. Προφανώς ένα Fog – Edge-computing σύστημα αποτελείται από πολλούς generators μικρής υπολογιστικής δύναμης, αφού μεριμνά τους είναι η συχνή παραγωγή δεδομένων . Οι aggregators είναι μικρότεροι σε αριθμό, αλλά υπολογιστικά δυνατότεροι αφού κάνουν μικρή προεπεξεργασία των δεδομένων. Τέλος οι processors, αλλιώς cloud-servers, είναι λίγοι σε αριθμό αλλα πολύ δυνατοί υπολογιστικά, αφού αποθηκεύουν και είναι σε θέση να επεξεργαστούν όλα τα δεδομένα του συστήματος.

Η εφαρμογή μας προσομοιώνει το Fog σύστημα για την λήψη και επεξεργασία θερμοκρασιών σε εργοστάσια. Η τοπολογία μας αποτελείται από εργοστάσια, καθένα εκ' των οποίων έχει παραμετροποιήσιμο αριθμό πατωμάτων και μηχανημάτων-θερμομέτρων σε αυτά. Έχουμε την δυνατότητα περαιτέρω παραμετροποίηση της τοπολογίας του συστήματος ορίζοντας των αριθμό των υποδικτύων και των aggregators ανά πάτωμα. Έτσι μπορούμε να πειραματιστούμε με εργοστάσια που έχουν περισσότερους συλλέκτες ανά πάτωμα που συνδέονται στο ίδιο υποδίκτυο ή περισσότερους συλλέκτες ανά πάτωμα που συνδέονται σε διαφορετικό υποδίκτυο και να μελετήσουμε τις διαφορές στα αποτελέσματα τους ως προς το delay μεταξύ των κόμβων, την συμφόρηση στα switches κλπ. Οι συσκευές, generators, σε κάθε πάτωμα του εργοστασίου παράγουν δεδομένα τα οποία συλλέγει ο aggregator του πατώματος. Τέλος κάθε εργοστάσιο έχει τον δικό του processor, όπου συλλέγονται τα δεδομένα και μπορούν να εμφανιστούν στην μορφή λίστας. Αναλυτικότερα οι generators προσομοιώνουν πραγματικές μετρήσεις θερμοκρασιών, οι οποίες προκύπτουν με hashing της περιοχής και του id του κάθε generator καθώς και από την χρονική στιγμή από την οποία λαμβάνεται η μέτρηση. Η χρονική στιγμή φροντίσαμε να επηρεάζει ημιτονοειδώς την θερμοκρασία, έτσι ώστε οι μεταβολές να είναι ομαλές και να μην έχουμε απότομες εναλλαγές κατά την διάρκεια της ημέρας. Η περίοδος λήψης των θερμοκρασιών μπορεί να είναι διαφορετική για κάθε generator. Ο κάθε generator στέλνει τις μετρήσεις του στον processor που του αντιστοιχεί, ο οποίος αναλαμβάνει την προεπεξεργασία των δεδομένων. Η κάθε μέτρηση περιλαμβάνει το id του generator, το floor του και ένα timestamp. Ο κάθε aggregator στέλνει με την σειρά του τους μέσους όρους που υπολογίζει στον processor, που είναι και ο τελικός προορισμός. Τα εργοστάσια συνδέονται σε κοινό internet πράγμα που επιτρέπει στο καθένα όποτε θέλει να έχει πρόσβαση στα δεδομένα του άλλου. Κάθε εργοστάσιο αποτελεί το δικό του δίκτυο, με τον processor ως κυριότερο κόμβο της ιεραρχίας να ανήκει στο κύριο δίκτυο και τους aggregators και generators να ανήκουν σε ξεχωριστά ή όχι υποδίκτυα ανάλογα με την τοπολογία. Τις τοπολογιες αυτες θα τις προσομοιώσουμε σε ενα fog computing emulation framework το Fogify. Αυτό μας επιτρέπει να δούμε εν λειτουργία το σύστημα, και να μελετήσουμε τα χαρακτηριστικά του. Έτσι γίνονται πιο ξεκάθαρες οι πιθανές αλλαγές και βελτιώσεις που μπορεί να χρειάζεται.

B. Εργαλεία που χρησιμοποιήθηκαν

Flask: Το Flask είναι ένα μικρό και ελαφρύ πλαίσιο web Python που παρέχει χρήσιμα εργαλεία και δυνατότητες που διευκολύνουν τη δημιουργία εφαρμογών ιστού στην Python. Κατατάσσεται ως μικροπλαίσιο επειδή δεν απαιτεί συγκεκριμένα εργαλεία ή βιβλιοθήκες. Δεν έχει επίπεδο αφαίρεσης βάσης δεδομένων, επικύρωση φόρμας ή άλλα στοιχεία όπου προϋπάρχουσες βιβλιοθήκες τρίτων παρέχουν κοινές λειτουργίες.

Docker: Το Docker (Ντόκερ) είναι μια πλατφόρμα λογισμικού ανοιχτού κώδικα που υλοποιεί εικονοποίηση (Virtualization) σε επίπεδο λειτουργικού συστήματος. Ουσιαστικά το Docker προσφέρει αυτοματοποιημένες διαδικασίες για την ανάπτυξη εφαρμογών σε απομονωμένες Περιοχές Χρήστη (User Spaces) που ονομάζονται Software Containers. Το λογισμικό χρησιμοποιεί τεχνολογίες του πυρήνα του Linux όπως τα cgroups και οι χώροι ονομάτων πυρήνα (kernel namespaces), για να επιτρέπει σε ανεξάρτητα software containers να εκτελούνται στο ίδιο λειτουργικό σύστημα. Έτσι αποφεύγεται η χρήση επιπλέον υπολογιστικών πόρων που θα απαιτούσε μια εικονική μηχανή (virtual machine).

Τα βασικά του τμήματα είναι:

- dockerfile: script που παρέχει πληροφορίες για το container που θα δημιουργηθεί

- image: αρχείο που χρησιμοποιείται για να εκτελεστεί ο κώδικας του container

- container: η "φούσκα" μέσα στην οποία τρέχουν οι εφαρμογές

- network: συνδέει τα containers μεταξύ τους

Docker Compose: Το Docker Compose είναι ένα εργαλείο που αναπτύχθηκε για να βοηθήσει στον καθορισμό και την κοινή χρήση εφαρμογών πολλαπλών κοντέινερ. Με το Compose, μπορούμε να δημιουργήσουμε ένα αρχείο YAML για να ορίσουμε τις υπηρεσίες και με μία μόνο εντολή, να τρέξουμε τα πάντα ή να τα καταστρέψουμε όλα.

Το μεγάλο πλεονέκτημα της χρήσης Compose είναι ότι μπορούμε να ορίσουμε τη στοίβα της εφαρμογής μας σε ένα αρχείο, να τη διατηρήσουμε στη βάση του πρότζεκτ μας και να επιτρέψουμε εύκολα σε κάποιον άλλο να συνεισφέρει στο έργο μας. Κάποιος θα χρειαστεί μόνο να κλωνοποιήσει το πρότζεκτ μας και να ξεκινήσει την εφαρμογή σύνθεσης.

Docker Swarm: Το Docker swarm είναι ένα εργαλείο ενορχήστρωσης κοντέινερ, που σημαίνει ότι επιτρέπει στο χρήστη να διαχειρίζεται πολλαπλά κοντέινερ που αναπτύσσονται σε πολλαπλούς κεντρικούς υπολογιστές. Ένα από τα βασικά πλεονεκτήματα που σχετίζονται με τη λειτουργία ενός docker swarm είναι το υψηλό επίπεδο διαθεσιμότητας που προσφέρεται για εφαρμογές.

C. Υλοποίηση

Χρησιμοποιήσαμε την γλώσσα προγραμματισμού Python για την δημιουργία των τριών services (generators, aggregators, processors). Για τον ορισμό του περιβάλλοντος του κάθε service χρησιμοποιήσαμε docker. Έτσι κάθε κόμβος τρέχει στο δικό του σύστημα-container με τα εργαλεία και τους πόρους που χρειάζεται. Για την δημιουργία των docker images για τα κάθε service χρησιμοποιήσαμε ως βάση το image "python:3-alpine", το οποίο περιλαμβάνει μια ελαφριά διανομή των linux με έτοιμο περιβάλλον Python.

Με το docker-compose συνδέσαμε τα containers μεταξύ τους και διαμορφώνοντας τα services σε ένα yaml file τα τρέχουμε ταυτόχρονα. Μέσω του yaml file καθορίζουμε την επικοινωνία μεταξύ των services και τις διάφορες παραμέτρους της εφαρμογής μας, όπως τα id και location με χρήση environment variables. Έτσι σε όποιον δώσουμε πρόσβαση στον κώδικα του project μας μπορεί να τρέξει στον υπολογιστή του παραδείγματα της εφαρμογής ακριβώς όπως τα τρέχουμε και εμείς. Για την επικοινωνία μεταξύ των services και την ανταλλαγή δεδομένων και αιτημάτων μεταξύ τους υλοποιήσαμε ένα API με χρήση των βιβλιοθηκών *flask* για τα endpoints και *requests* για την αποστολή μηνυμάτων. Τα requests χρησιμοποιούν TCP/IP πρωτόκολλο και επομένως η επικοινωνία στα endpoints είναι σύγχρονη.

Η εφαρμογή μας είναι παραμετροποιήσιμη μέσω των εξής μεταβλητων περιβαλλοντος για τα services:

INTERVAL: η συχνότητα λήψης των μετρήσεων από κάθε generator σε δευτερόλεπτα

WINDOW: το πλήθος των μετρήσεων που επεξεργάζονται οι aggregators για κάθε πακέτο

HISTORY_LENGTH: το πλήθος των πακέτων που διατηρούνται στη μνήμη των processor για κάθε aggregator

BUSY_WAIT: οι επιπλέον κύκλοι μηχανής που εκτελούν οι aggregator πριν την αποστολή κάθε πακέτου

Η διεπαφή με τον χρήστη της εφαρμογής γίνεται μέσω των endpoint του processor. Μέσω αυτών, παρέχουμε πρόσβαση στο ιστορικό των μετρήσεων θερμοκρασίας, καθώς και σε διαγνωστικά στοιχεία τα οποία κρίναμε χρήσιμα για να μας βοηθήσουν κατά την διεξαγωγή των πειραμάτων της εφαρμογής.

/history
Το ιστορικό όλων των πακέτων

/latest
Το πιο πρόσφατο πακέτο

/maximum
Το πακέτο με την μεγαλύτερη θερμοκρασία εντός ιστορικού

/delay
Χρόνος που μεσολαβεί για να φτάσει το τελευταίο μήνυμα που έφυγε από τον generator στον processor (μέσος όρος όλων των πακέτων του ιστορικού)

/computation
Χρόνος επεξεργασίας κάθε πακέτου πριν αποσταλεί στον processor (μέσος όρος όλων των πακέτων του ιστορικού)

/span
Χρόνος που μεσολαβεί για να φτάσει το πρώτο μήνυμα που έφυγε από τον generator στον processor (μέσος όρος όλων των πακέτων του ιστορικού)

/diagnose
Συνδυασμός των delay, computation και span.

Κάθε endpoint μπορεί να απαντήσει ερωτήματα είτε για το σύνολο όλων των generator του εργοστασίου, είτε για συγκεκριμένους generators ανάλογα με το id ή το location τους. Τα ερωτήματα γίνονται στην εξής μορφή:

/query/all
Επιστρέφει δεδομένα για όλους τους generators του εργοστασίου

/query?id=X&id=Y
Επιστρέφει δεδομένα για τους generators με βάση το id τους

/query?location=X&location=Y
Επιστρέφει δεδομένα για τους generators με βάση την τοποθεσία τους (όροφο εργοστασίου)

όπου query είναι ένα από τα προηγουμένως ορισμένα endpoint.

D. Παράδειγμα εφαρμογής με docker compose

Ένα παράδειγμα της εφαρμογής με docker compose είναι να τρέξουμε το σύστημα με δύο generators, ένα aggregator και ένα processor ("unfogified.yaml" στον κώδικα). Έτσι βλέπουμε πως δουλεύει το σύστημά μας για ένα εργοστάσιο με τις παραπάνω υπηρεσίες. Κάνοντας hit στα endpoints που αναφέραμε μπορούμε να δούμε τα αποτελέσματα στην μορφή λίστας. Έτσι βλέπουμε τις θερμοκρασίες των μηχανημάτων του εργοστασίου μας αφού έχουν δεχτεί την προεπεξεργασία των aggregators. Τι γίνεται όμως με τα χαρακτηριστικά του δικτύου και των υποδικτύων του συστήματος; Τι γίνεται με τις καθυστερήσεις και τις συμφόρηση μεταξύ των κόμβων του συστήματος και των switches; Και πως θα μπορέσουμε να σηκώσουμε ένα σύστημα με περισσότερους υπολογιστικούς πόρους από ότι διαθέτει ο υπολογιστής μας.

III. ΠΡΟΣΟΜΟΙΩΣΗ ΚΑΙ ΑΝΑΠΤΥΞΗ ΤΟΠΟΛΟΓΙΩΝ

A. Εισαγωγή

Όπως είδαμε παραπάνω στο παράδειγμα εφαρμογής με docker compose οι δυνατότητες που μας δίνονται είναι περιορισμένες και δεν έχουμε την πλήρη εικόνα λειτουργίας του συστήματός μας. Σε περίπτωση που θέλουμε να μοντελοποιήσουμε ένα σύστημα, όπως στο παράδειγμά μας ένα εργοστάσιο, θα θέλουμε να είμαστε σε θέση να δοκιμάσουμε, πολλές τοπολογίες, να αντλήσουμε τα χαρακτηριστικά τους, να τα συγκρίνουμε μεταξύ τους και να συμπεράνουμε ποια από αυτές είναι η κατάλληλη για το σύστημά μας και όλα αυτά εύκολα και γρήγορα. Επομένως για τον έλεγχο λειτουργίας χρειάζεται ένα λογισμικό που επιτρέπει την προσομοίωση του συστήματος. Ένα τέτοιο λογισμικό είναι το Fog Computing Emulation Framework: Fogify.

Το **Fogify** [1] είναι ένα πλαίσιο εξομοίωσης που διευκολύνει τη μοντελοποίηση, την ανάπτυξη και τον πειραματισμό των πλατφορμών ομίχλης. Παρέχει ένα σύνολο εργαλείων για:

- μοντελοποίηση σύνθετων τοπολογιών ομίχλης που αποτελούνται από ετερογενείς πόρους, δυνατότητες δικτύου και κριτήρια QoS

- εκτέλεση και ενορχήστρωση των μοντελοποιημένων υπηρεσιών χρησιμοποιώντας προδιαγραφές υποδομής με κώδικα σε μορφή κοντέινερ τοπικά ή στο cloud μέσω του docker-swarm

- πειραματισμό, μέτρηση και αξιολόγηση κάθε σεναρίου εισάγοντας σφάλματα και προσαρμόζοντας τη διαμόρφωση κατά το χρόνο εκτέλεσης για να δοκιμάσετε διαφορετικά σενάρια "what-if" που αποκαλύπτουν τους περιορισμούς μιας υπηρεσίας πριν παρουσιαστούν στο κοινό.

Η εκτέλεση των προσομοιώσεων γίνεται μέσω **Jupyter** notebook, μιας διαδραστικής διεπαφής για ανάλυση δεδομένων. Η περιγραφή του κάθε σεναρίου προσομοίωσης του fogify ορίζεται μεσω των services, networks, nodes και topology που την αποτελούν. Αυτά θα τα παράγουμε από ένα dictionary που τα περιέχει σε μορφή λιστών, το οποίο εξάγεται σαν ένα yaml αρχείο (fogified docker-compose). Έτσι θα μπορούμε να δεχτούμε οποιαδήποτε τοπολογία και να την μεταφράσουμε στην fogified μορφή της.

Η σύνδεση των υπολογιστικών κόμβων μεταξύ τους είναι κουραστικό να γίνεται χειροκίνητα και σίγουρα μη κλιμακώσιμο. Για αυτό κάνουμε χρήση ενός λογισμικού παραγωγής τοπολογιών, που μας επιτρέπει να περιγραψουμε την τοπολογία με απλές δομές και να αλλάξουμε τους υπολογιστικούς κόμβους εύκολα και γρήγορα.

Το **Ether** [2] είναι ένα εργαλείο Python για τη δημιουργία εύλογων διαμορφώσεων υποδομής άκρων. Προέκυψε από την ανάγκη αξιολόγησης συστημάτων υπολογιστών αιχμής -edge computing systems- σε διαφορετικά σενάρια υποδομής όπου δεν υπάρχουν διαθέσιμες κατάλληλες δοκιμαστικές πλατφόρμες.

Μερικές από τις περιπτώσεις χρήσεων για το ether είναι:

- Αξιολόγηση στρατηγικών κατανομής πόρων

- Δημιουργία τοπολογιών για προσομοιώσεις δικτύου

- Προγραμματισμός χωρητικότητας υποδομής

B. Γεννήτρια Τοπολογιών

Κατασκευάσαμε το δικό μας σενάριο υποδομής χρησιμοποιώντας αντικειμενοστραφή προγραμματισμό και επεκτείνοντας τις κλάσεις του Ether. Συγκεκριμένα, ορίσαμε οντότητες για πόλεις, εργοστάσια και ορόφους, με την τοπολογία μας να μπορεί να περιλαμβάνει πολλαπλά εργοστάσια ανά πόλη και πολλούς ορόφους ανά εργοστάσιο. Κάθε εργοστάσιο έχει ένα κεντρικό δίκτυο, καθώς και επιπλέον ένα δίκτυο ανά όροφο, τα οποία επικοινωνούν με το κεντρικό μέσω switches. Το κεντρικό δίκτυο κάθε εργοστασίου περιλαμβάνει τον processor της εφαρμογής μας, ενώ το δίκτυο κάθε ορόφου μπορεί να περιλαμβάνει μεταβλητό αριθμό από aggregators και generators. Τα εργοστάσια συνδέονται με το δίκτυο κάθε πόλης και – μέσω αυτού – με τον παγκόσμιο ιστό.

Η παραμετροποίηση του σεναρίου γίνεται με χρήση αρχείου εισόδου σε μορφή yaml. Εκεί ορίζονται ιεραρχικά οι οντότητες που θέλουμε να συμπεριλάβουμε στην τοπολογία μας. Όλες οι οντότητες ορίζονται ονομαστικά εκτός από τους generators, το πλήθος των οποίων ορίζεται ως παράμετρος του κάθε aggregator που τους διαχειρίζεται. Κάθε δίκτυο χαρακτηρίζεται από τις καθυστερήσεις επικοινωνίας των κόμβων του, οι οποίες δίνονται σε μορφή λογαριθμικής κατανομής. Εκτός της τοπολογίας, ορίζουμε και τους πόρους συστήματος που θέλουμε να αποδώσουμε σε κάθε είδος κόμβου, δηλαδή τον αριθμό των πυρήνων, την ταχύτητα του ρολογιού ανά πυρήνα και τη διαθέσιμη μνήμη για κάθε processor, aggregator ή generator. Τέλος, ορίζουμε τις καθολικές παραμέτρους της εφαρμογής μας (history, window, interval, busy_wait).

Κατά την κατασκευή κάθε κόμβου από το Ether αποθηκεύουμε όλες τις παραμέτρους που απαιτούνται από την εφαρμογή μας χρησιμοποιώντας ετικέτες (labels). Αυτές περιλαμβάνουν τις καθολικές παραμέτρους της εφαρμογής, αλλά και μεταβλητές περιβάλλοντος που επιβάλλονται από την τοπολογία. Για παράδειγμα, με κάθε generator αποθηκεύουμε και τον aggregator κόμβο στον οποίο θα αποστέλλει τις μετρήσεις του. Για τον προσδιορισμό των πόρων συστήματος για κάθε κόμβο, επεκτείναμε την υπεύθυνη κλάση του Ether, η οποία δεν περιλαμβάνει μεταβλητό πλήθος πυρήνων για κάθε κόμβο, κάτι το οποίο έχει αξία να διαθέσουμε στο Fogify. Προαιρετικά, ορίζουμε την θύρα του host στην οποία θα είναι διαθέσιμο του API κάθε processor κατά την προσομοίωση.

Αφού δημιουργηθεί η τοπολογία του Ether με βάση του σεναρίου της εφαρμογής και του αρχείου περιγραφής της υποδομής, θα πρέπει να γίνει εξαγωγή της σε μορφή “fogified docker compose” ώστε να προσομοιωθεί από το Fogify. Καθώς το Fogify δεν επιτρέπει τον ορισμό μεταβλητών περιβάλλοντος ανά κόμβο, αλλά μόνο ανά service, θα πρέπει αρχικά να δημιουργήσουμε ένα service για κάθε κόμβο. Κάθε service συμπληρώνεται με βάση τα labels που έχουν αποθηκευτεί στον κόμβο από το Ether, καθώς και από τις βασικές απαιτήσεις της εφαρμογής μας, όπως οι θύρες επικοινωνίας των services. Έπειτα ορίζουμε τους πόρους που θα διατεθούν σε κάθε κόμβο με απλή μετατροπή μονάδων. Για την προσομοίωση των συνδέσεων μεταξύ των κόμβων, ορίζουμε καθολικό δίκτυο σε επίπεδο Fogify το οποίο θα περιλαμβάνει όλους του κόμβους της τοπολογίας. Για κάθε ζεύγος κόμβων ορίσαμε ένα “network link”, το οποίο ορίζει την συμπεριφορά της επικοινωνία μεταξύ αυτών με βάση τις παραμέτρους delay και bandwidth. Ο προσδιορισμός αυτών γίνεται με κλήσεις στο API του Ether για το μονοπάτι (route) που συνδέει κάθε ζεύγος κόμβων. Το delay βρίσκεται ως το μισό του roundtrip time του μονοπατιού, ενώ το bandwidth ως το ελάχιστο bandwidth του μονοπατιού. Τελικώς, ορίζουμε το σχηματικο τοπολογιών (topology blueprints) προς προσομοίωση, συσχετίζοντας τους κόμβους και τους πόρους τους με τα παραμετροποιημένα services, συνδέοντας το σύνολό των blueprint στο δίκτυο και ζητώντας 1 replica για την κάθε blueprint.

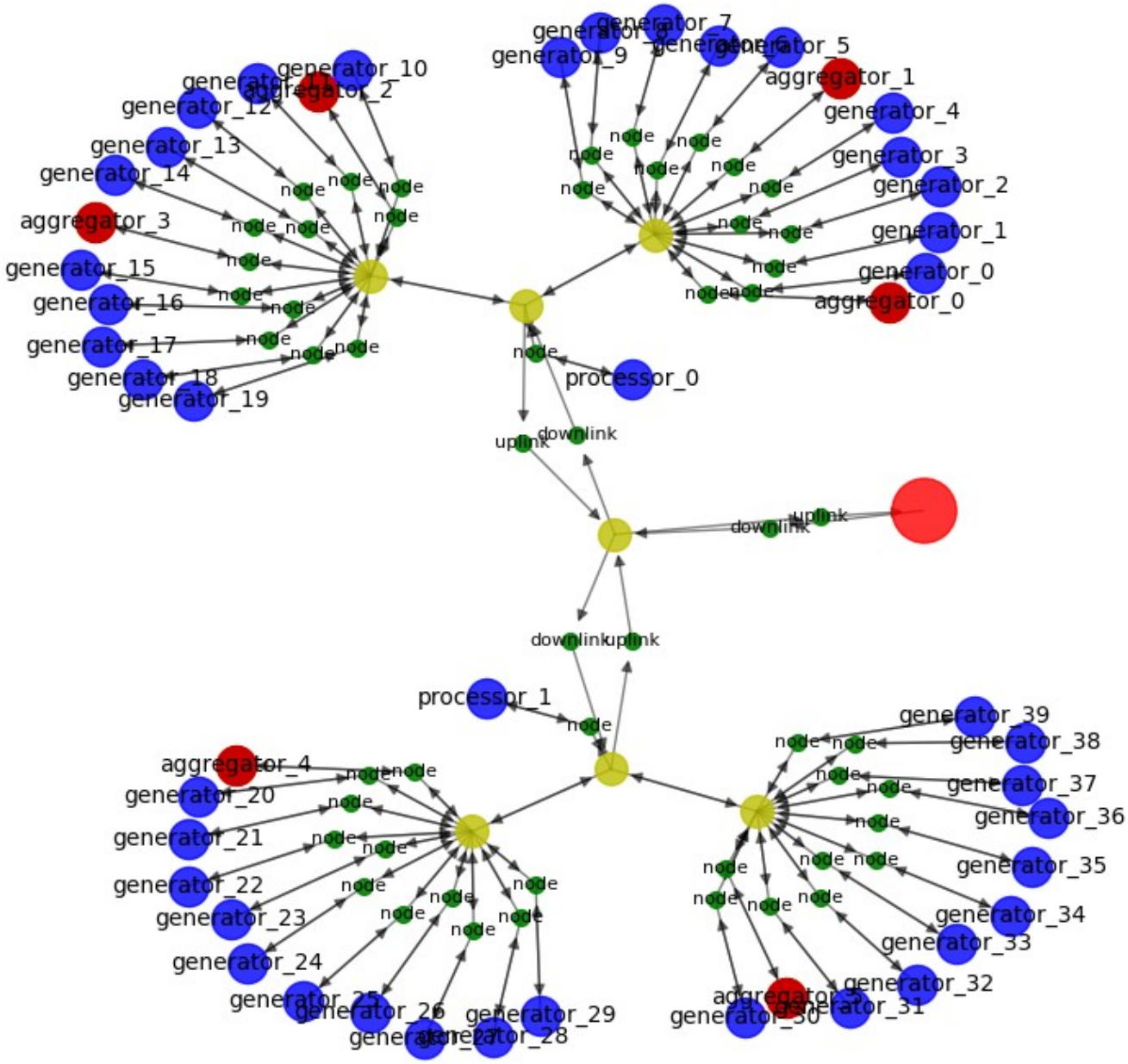
Για την εκτέλεση της γεννήτριας τοπολογιών διαθέτουμε διεπαφή τερματικού, όπου το αρχείο εισόδου δίνεται ως πρώτη και μοναδική παράμετρος. Η τοπολογία απεικονίζεται γραφικά με χρήση των εργαλείων του Ether για εποπτεία από τον χρήστη, ενώ η περιγραφή της τοπολογίας – έτοιμη για προσομοίωση από το Fogify – γράφεται στην έξοδο του τερματικού.

C. Παράδειγμα Εφαρμογής με Fogify

Ένα παράδειγμα της εφαρμογής με fogify είναι να μοντελοποιήσουμε ένα σύστημα που περιέχει δύο διαφορετικά εργοστάσια, το καθένα με την δική του τοπολογία, έτσι ώστε να τα συγκρίνουμε και να δούμε πιο είναι κατάλληλο για εμάς. Και τα δύο εργοστάσια διαθέτουν δύο floors με 10 generators στο καθένα. Το πρώτο εργοστάσιο διαθέτει 2 aggregators σε κάθε floor, ενώ το δεύτερο διαθέτει 1 aggregator σε κάθε floor. Το πρώτο floor

κάθε εργοστασίου είναι υποδίκτυο LAN, επομένως έχει μικρότερο delay στα links μεταξύ των κόμβων, ενώ το δεύτερο floor τους είναι υποδίκτυο WAN και έχει μεγαλυτερο delay μεταξύ των κόμβων. Τρέχοντας την παραπάνω τοπολογία μπορούμε να συμπεράνουμε ποια τοπολογία εργοστασίου δίνει τα καλύτερα αποτελέσματα ως προς την καθυστέρηση και την συμφόρηση μεταξύ των κόμβων, αν αξίζει να έχουμε δύο aggregators σε κάθε floor αντί για ένα και πως συμπεριφέρεται το κάθε σύστημα ανάλογα με τον τρόπο σύνδεσης των κόμβων στο υποδίκτυο.

Η τοπολογία της παραπάνω εφαρμογής μπορεί για διευκόλυνσή μας να παραχθεί από το ether, το μόνο που θα χρειαστεί θα είναι η μετάφραση της τοπολογίας σε fogified μορφή, όπως περιγράψαμε στις προηγούμενες ενότητες. Παρακάτω φαίνεται η τοπολογία του παραδείγματος παραγμένη από το ether.



D. Οδηγίες για το τρέξιμο της εφαρμογής

Δεδομένου ότι έχει γίνει clone ο κώδικας της εργασίας και έχουν εγκατασταθεί σωστά τα Fogify και Ether:

Αρχικά πρέπει να γίνει build των docker images:

```
$ cd asps22
$ make all
```

Έπειτα ελευθερώνουμε το swarm:

```
$ docker swarm leave –force
```

Βρίσκουμε την ip μας:

```
$ ifconfig
```

Και θέτουμε την ip μας σαν manager του swarm:

```
$ docker swarm init --advertise-addr [ip]
```

Φτιάχνουμε την τοπολογία μέσω του ether τρέχοντας το αρχείο fogify_example.yml με την παρακάτω εντολή

```
$ python3 src/topology/main.py \
examples/fogify_example.yml > \
examples/fogified/fogify_example.out.yml
```

Μας εμφανίζεται στην οθόνη η τοπολογία σε εικόνα και κλείνοντάς την δημιουργείται το αντίστοιχο αρχείο output.yml

Σε μια άλλη κονσόλα μπαίνουμε στον φάκελο του fogify-demo και ανοίγουμε τον προσομοιωτή τοπολογιών του fogify εκτελώντας

```
$ docker-compose -p fogemulator up
```

Ανοίγουμε το jupiter στο link: http://127.0.0.1:8888/lab

Μετά πρέπει να ανέβουν στο Jupyter του Fogify τα αρχεία:

- examples/run.ipynb
- examples/fogified/fogify_example.out.yml

Το deployment γίνεται από Jupyter με προσαρμογή της παραμέτρου της εντολής:

```
fogify = FogifySDK("http://controller:5000", \
"fogify_example.out.yml")
```

Όταν εκτελέσουμε το deploy ελέγχουμε αν έχουν ανέβει όλα τα services (replicas 1/1) με την εντολή

```
$ docker service list
```

Αφού γίνει κάθε τοπολογία deploy συνήθως χρειάζονται δύο λεπτά για να φτάσει σε κατάσταση ισορροπίας το σύστημα. Κάθε πακέτο που φτάνει στον server περιέχει τον μέσο όρο θερμοκρασιών 10 μετρήσεων ενός σένσορα (window), καθώς και κάποια διαγνωστικά. Ο server κρατάει στην μνήμη τα τελευταία 10 πακέτα (history) για κάθε σένσορα. Επιλέχθηκε μικρό ώστε να φτάνουμε ταχύτερα σε κατάσταση ισορροπίας στο endpoint.

Τα endpoints του processor προωθούνται στο localhost και εκεί μπορούμε να δούμε τα αποτελέσματα, οπότε τα διαγνωστικά που θέλουμε για κάθε πείραμα θα είναι διαθέσιμα στη url:

http://127.0.0.1:5005/diagnose/all

Για να τρέξουμε την επόμενη τοπολογία τρέχουμε την εντολή undeploy() του jupiter και επαναλαμβάνουμε την διαδικασία.

IV. ΠΕΙΡΑΜΑΤΑ

A. Περιγραφή πειραμάτων

Τα configuration που δοκιμάσαμε για την διεξαγωγή των πειραμάτων είναι τα εξής:

simple-lan: βασική τοπολογία με 10 edge devices, 1 aggregator, 1 server. 10 μετρήσεις ανά δευτερόλεπτο ανά σένσορα, aggregation με window 10, server history 10

simple-wan: το ίδιο με simple-lan, αλλά με χειρότερο latency στο δίκτυο, οπότε αναμένουμε μεγαλύτερο "delay"

loaded-bundled: το ίδιο με simple-lan, αλλά με 100 μετρήσεις ανά δευτερόλεπτο και aggregation με window 100. Ενώ θεωρητικά έχουμε πάλι 1 aggregation το δευτερόλεπτο, θα πρέπει να παρατηρήσουμε αρκετά μεγαλύτερο "span" λόγω latency

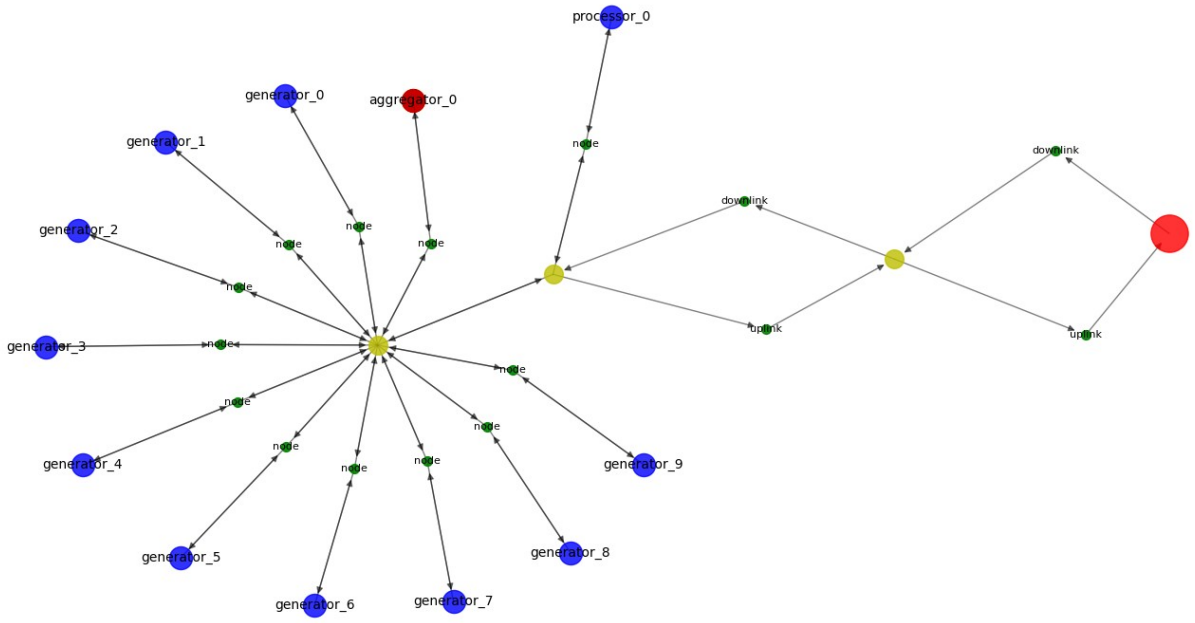
loaded-aggregators: το ιδιο με loaded-bundled, άλλα σπάμε τους generators σε δύο group των 5 πάνω στο ίδιο δίκτυο, κάθε group με δικό του aggregator. θεωρητικά καλύτερο "span" και "delay" αφού μοιράζονται τα trafic και computation time

loaded-plenty: το ίδιο με loaded-aggregators, άλλα δίνουμε παραπάνω υπολογιστικούς πόρους στους aggregators

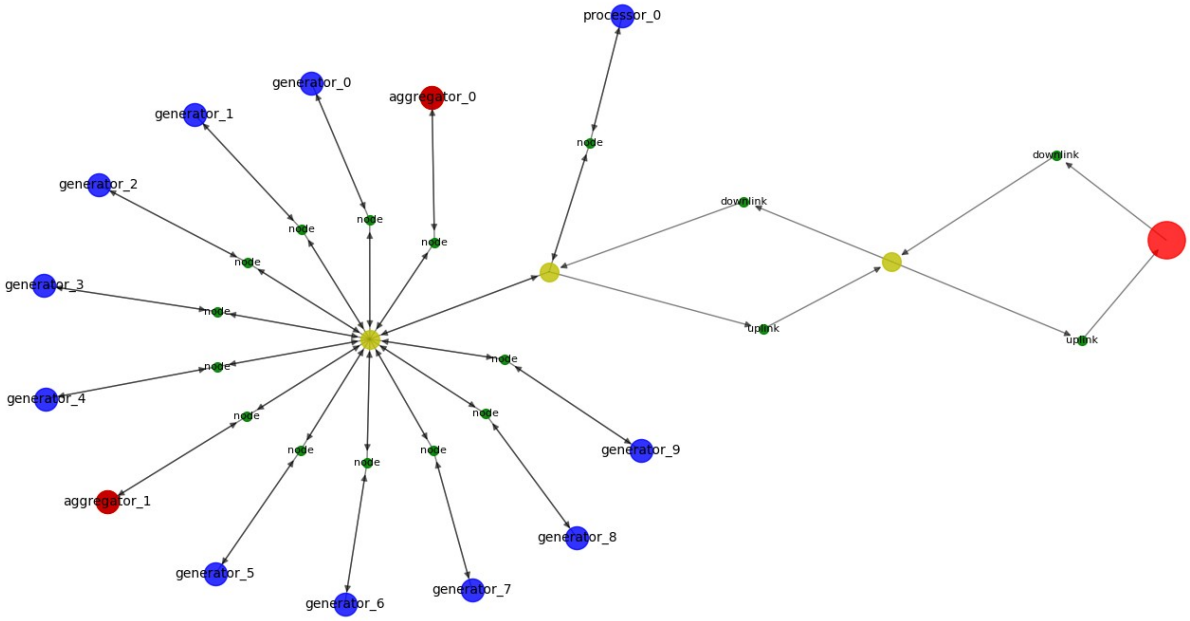
Τα configuration έγιναν με τέτοιο τρόπο που τα delay εντός lan είναι τα ίδια για κάθε κόμβο, με σκοπό να αναδείξουμε τα οφέλη των τοπολογιών. Στο simple-wan τα delay ακολουθούν λογαριθμική κατανομή.

Πείραμα	Computation	Delay	Span
simple-lan	0.900	1.641	8.302
simple-wan	1.195	2.015	10.426
loaded-bundled	1.729	3.294	50.973
loaded-aggregators	1.790	2.468	25.191
loaded-plenty	0.280	1.199	8.044

Αποτελέσματα πειραμάτων (μονάδες χρόνου: sec)



topology with 10 generators, 1 aggregator, 1 processor



topology with 10 generators, 2 aggregators, 1 processor

Β. Παρατηρήσεις:

Το simple-lan έχει μικρότερες τιμές στα αποτελέσματα από ότι το simple wan όπως αναμέναμε, αφού μια ethernet σύνδεση θα έχει μικρότερο delay από μια wireless σύνδεση.

Το loaded-bundled μας δίνει όντως μεγαλύτερο span, το οποίο οφείλεται στο ότι αυξήσαμε τις μετρήσεις ανά δευτερόλεπτο και επειδή οι επικοινωνίες που χρησιμοποιούμε είναι σύγχρονες θέλουμε περισσότερο χρόνο στην επικοινωνία μεταξύ των κόμβων για να γίνουν τα acknowledgements και επομένως έχουμε μεγαλύτερη συμφόρηση.

Στο loaded-aggregators πετυχαίνουμε καλύτερους χρόνους σε σχέση με το loaded-bundled, το οποίο αποδεικνύει ότι όταν μοιράζουμε την δουλειά ενός floor σε περισσότερους aggregators παίρνουμε καλύτερα αποτελέσματα.

Στο loaded-plenty, όπου έχουμε πιο ισχυρούς υπολογιστικούς πόρους στους aggregators από το loaded-aggregators, παίρνουμε καλύτερα αποτελέσματα το οποίο αναμέναμε αφού μειώνεται σημαντικά το computation time, που αποτελεί παράγοντα για την μείωση του delay και συνεπώς του span.

ΕΠΙΛΟΓΟΣ

Συμπεραίνουμε ότι η χρήση λογισμικού για την κατασκευή και την προσομοίωση τοπολογιών διευκολύνει σημαντικά την διαδικασία σχεδίασης και μοντελοποίησης συστημάτων αιχμής. Για παράδειγμα, μπορέσαμε να περιγράψουμε πλήρως την τοπολογία του παραδείγματος simple-lan σε αρχείου 37 γραμμών, ενώ αντίστοιχο fogified docker-compose file είναι μήκους 721 γραμμων. Τέλος, παρατηρούμε ότι τα Quality of Service χαρακτηριστικά του δικτύου και ο αριθμός τον aggregator παίζουν σημαντικό ρόλο στην απόδοση του συστήματος. Έχει λοιπόν αξία η χρήση γεννήτορων τοπολογιών και τεχνολογιών προσομοίωσης για τον σωστό σχεδιασμό τοπολογιών συστημάτων αιχμής.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] *Fogify: A Fog Computing Emulation Framework. Symeonides, M., Georgiou, Z., Trihinas, D., Pallis, G., & Dikaiakos, M. In Proceedings of the 5th ACM/IEEE Symposium on Edge Computing, of SEC '20, New York, NY, USA, 2020. Association for Computing Machinery.*

[2] *Synthesizing Plausible Infrastructure Configurations for Evaluating Edge Computing Systems: Rausch, T., Lachner, C., Frangoudis, P. A., Raith, P., & Dustdar, S. (2020). In 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20). USENIX Association.*

GitHub links for dependencies:

Fogify: https://github.com/UCY-LINC-LAB/fogify

Ether: https://github.com/edgerun/ether