Ενσωματωμένα συστήματα πραγματικού χρόνου

Τελικό Project - Συλλογή δεδομένων για χρηματοοικονομικά μέσα

Θεόδωρος Κατζάλης
ΑΕΜ: 9282
katzalis@ece.auth.gr

Ιούνιος 2024

Εισαγωγή

Η εργασία πραγματεύεται την ασύγχρονη συλλογή πληροφορίας συναλλαγών για σύμβολα από την πλατφόρμα finnhub. Τα σύμβολα που μελετήσαμε είναι AMZN, MSFT, BINANCE:USDT, και IC MARKETS: 1.

Η υλοποίηση αποσκοπεί στην αδιάκοπη λειτουργία του προγράμματος σε ένα ενσωματωμένο σύστημα, χρησιμοποιώντας παράλληλα νήματα και πιο συγκεκριμένα βασιζόμενοι στο μοντέλο συγχρονισμού producer-consumer.

Οι λειτουργίες που μας απασχολούν είναι:

- Καταγραφή συναλλαγών
- Υπολογισμός candlestick για κάθε λεπτό, στο λεπτό
- Υπολογισμός κινητού μέσου όρου των τιμών (price) των συμβόλων για τα πιο πρόσφατα 15 λεπτά

Καθένα απο τα παραπάνω αποθηκεύεται σε αρχείο ξεχωριστό για το κάθε σύμβολο. Το πρόγραμμα υλοποιήθηκε σε C, με pthreads για παραλληλοποίηση και με την χρήση της βιβλιοθήκης libwebsocket για την δημιουργία websocket ως το αμφίδρομο κανάλι επικοινωνίας με την πλατφόρμα finnhub.

Η εργασία και ο κώδικας μπορούν να βρεθούν σε αυτό το αποθετήριο https://github.com/thodkatz/embedded-rtos. Ο φάκελος `producer_consumer` περιέχει ένα πρότυπο πρόβλημα producer-consumer (preparatory homework), και ο φάκελος `finnhub` την συγκεκριμένη εργασία, βασιζόμενοι σε αυτό το πρότυπο. Το πρόγραμμα είχε χρόνο εκτέλεσης 50:46:18.77 (hh:mm:ss).

Μεθοδολογία

Πρωταρχικό στάδιο ήταν η υλοποίηση ενός finnhub client με websocket, και η επιτυχής λήψη συναλλαγών. Αρχικά, μέσω του websocket στείλαμε μηνύματα εγγραφής στα σύμβολα που μας απασχολούν, όπως ορίζεται και από το API documentation (π.χ `'{"type":"subscribe-news","symbol":"AAPL"}'`), και στη συνέχεια λαμβάναμε την πληροφορία

των συναλλαγών με callback συναρτήσεις. Η αφετηρία της λήψης πληροφορίων γίνεται με την κλήση της `lws_service()`, υπεύθυνη για την διαχείριση των events.

Αυτό που θα μας απασχολήσει περισσότερο είναι η τεχνική παραλληλοποίησης του προβλήματος. Όπως αναφέρθηκε προηγουμένως, η προσέγγιση μας βασίζεται στο πρόβλημα producer-consumer. Οι producers είναι υπεύθυνοι να περιμένουν και να προσθέτουν την πληροφορία των events (`queueAdd()`) σε μια κοινή ουρά. Οι consumers, σε πραγματικό χρόνο, καταναλώνουν τα δεδομένα απο την ουρά (`queueDel()`), καταγράφοντας σε αρχείο τις συναλλαγές, και υπολογίζοντας τις αθροιστικές τιμές για candlesticks και για τον κινητό μέσο όρο.

Για την καταγραφή των candlesticks στο λεπτό και του κινητού μέσου όρου, έχουμε ένα ξεχωριστό νήμα, τον scheduler. Αυτό παραμένει αδρανές για ένα λεπτό, μέχρι οι producers και οι consumers να συλλέξουν και επεξεργαστούν τα δεδομένα, και στη συνέχεια αποθηκεύει τους υπολογισμούς σε αρχεία. Όλα τα νήματα ασχολούνται με κοινούς πόρους και για την επίτευξη συνοχής διαμοιράζονται ένα mutex. Έτσι, οταν καλείται ο scheduler και γίνεται κάτοχος του κλειδιού (η κοινή ουρά παύει να υφίσταται οποιαδήποτε επεξεργασία απο τους producers και consumers), καταγράφουμε σε αρχεία τις πληροφορίες που συλλέχθηκαν στο μεσοδιάστημα, και αρχικοποιούνται οι δομές δεδομένων (`struct candlestickMinute`) υπεύθυνες για τους υπολογισμούς του ενός λεπτού.

Ο scheduler ακόμη, αποθηκεύει την μέση τιμή της τιμής (price) των συναλλαγών, κατά τη διάρκεια ενός λεπτού ως την τιμή του λεπτού, στην δομή δεδομένων που είναι υπεύθυνη για τον υπολογισμό του κινητού μέσου όρου (`struct movingAverage`), που αποτελείται απο μια λίστα 15 στοιχείων, των πιο πρόσφατων λεπτών.

Επομένως υπάρχουν αλληλεξαρτήσεις στα νήματα που έχουμε αναφέρει. Οι producers προσθέτουν στην ουρά, οι consumers αφαιρούν απο την ουρά, και ο scheduler "παγώνει" την επεξεργασία πληροφορίας για να καταγράψει την κατάσταση του λεπτού. Ενδιαφέρον παρουσιάζει, η καθυστέρηση του scheduler εξαιτίας του mutex, και του ανταγωνισμού με τα υπόλοιπα νήματα στο να αποκτήσει το κλειδί. Για να αυξήσουμε την ακρίβεια ως προς τον χρόνο, για την καταγραφή των δεδομένων, θα μπορούσαμε να θέσουμε το νήμα του scheduler με την υψηλότερη προτεραιότητα, αλλά θα έπρεπε να εξεταστεί και η επεξεργασία των εναπομείναντων δεδομένων της κοινής ουράς από τους consumers. Οπότε η βέλτιστη πολιτική προτεραιότητας θα ήταν κατά την κλήση του scheduler, να "παγώνουν" οι producers, να αδειάζουν την ουρά οι consumers, και ύστερα να γίνεται η καταγραφή.

Η παραπάνω διαπίστωση έγινε σε μεταγενέστερο στάδιο και τα αποτελέσματα μας ακολουθούν μια πιο απλή μεθοδολογία, όπου όλα τα νήματα έχουν την ίδια προτεραιότητα, και ο scheduler, όταν καλείται, μετά την απόκτηση του κλειδιού, καταγράφει την πληροφορία χωρίς κάποια συνθήκη. Το μειονέκτημα είναι ότι υπάρχει η πιθανότητα να μην συμπεριλάβουμε στους υπολογισμούς κάποιες συναλλαγές, που ανήκουν στο τρέχων λεπτό, και δεν έχουν επεξεργαστεί απο τους consumers όταν κλήθηκε ο scheduler. Ωστόσο η πληροφορία δεν χάνεται και προσμετράται για το επόμενο λεπτό, στην επόμενη κλήση του scheduler.

Για την εξασφάλιση της αδιάκοπης λειτουργίας, οι producers είναι υπεύθυνοι για την επανασύνδεση με το websocket, και το πρόγραμμα τερματίζει με την αποστολή σήματος SIGINT.

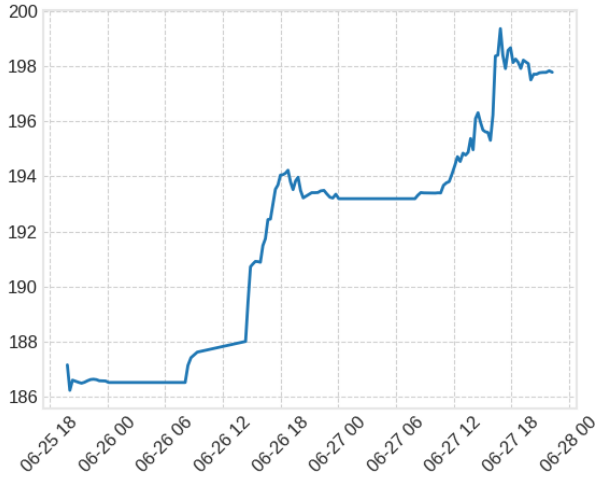
Αξίζει να σημειωθεί ότι χρησιμοποιήθηκαν 2 νήματα για producers, 2 για consumers, και 1 scheduler. Το μέγεθος της κοινής ουράς ήταν 100 στοιχείων. Η μελέτη για την επίδραση της ουράς, και τον αριθμό των νημάτων έχει γίνει στο prepatory homework, του οποίου την αναφορά μπορείτε να δείτε εδώ.

Αποτελέσματα

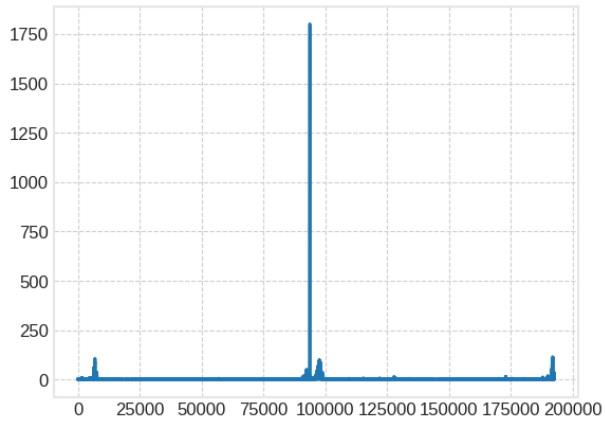
Παρακάτω παραθέτουμε τα αποτελέσματά του προγράμματος. Μπορούμε να δούμε την α) πορεία των candlesticks, β) τον κινητό μέσο όρο της τιμής των συμβόλων για τα 15 πιο πρόσφατα λεπτά, και γ)

το χρόνο που μεσολάβησε μεταξύ του χρόνου καταγραφής της συναλλαγής με την χρονική στιγμή της συναλλαγής που παρέχεται ως πεδίο των δεδομένων του finhhub. Ακόμη, για να δούμε πόσο αποκλίνει ο scheduler απο το λεπτό, όπως αναφέρθηκε προηγουμένως ορίσαμε ως ποσοστιαία απόκλιση του scheduler: $\frac{\text{Χρονική στιγμή καταγραφής scheduler} - (\text{Προηγούμενη χρονική στιγμή καταγραφής scheduler} + 1 \text{ λεπτό})}{1 \text{ λεπτό}}$

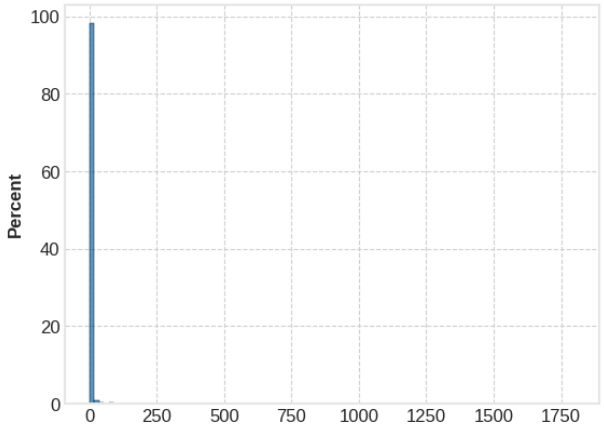
(a) Candlesticks



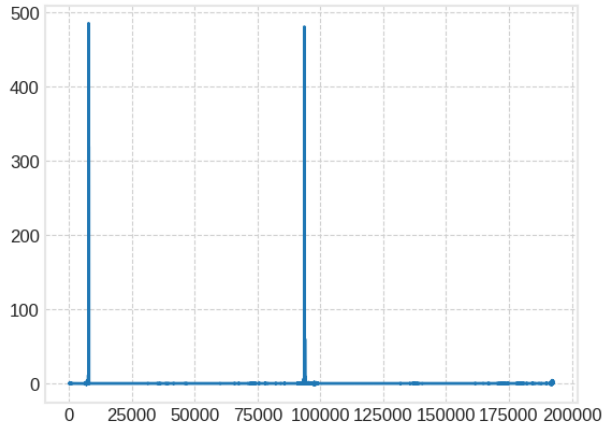
(b) Κινητός μέσος όρος τιμής 15 πιο πρόσφατων λεπτών



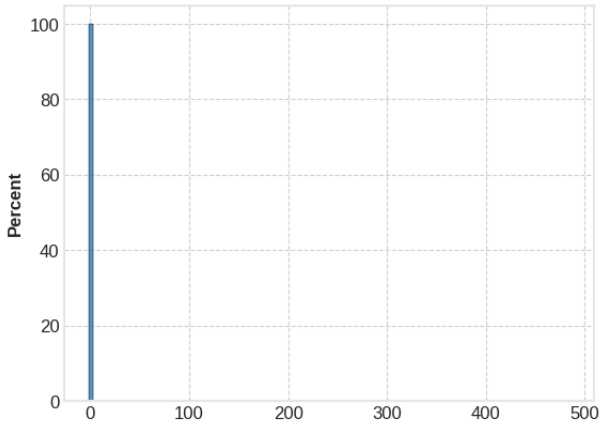
(c) Χρονική καθυστέρηση καταγραφής συναλλαγών (s)



(d) Ιστόγραμμα χρονικής καθυστέρησης καταγραφής συναλλαγών (s)



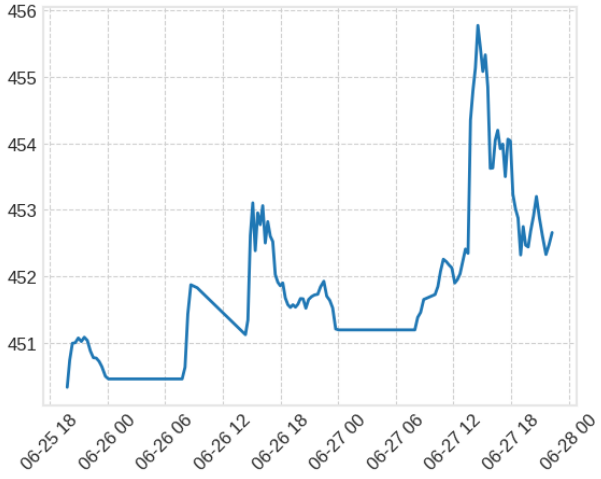
(e) Ποσοστιαία απόκλιση του λεπτού του scheduler



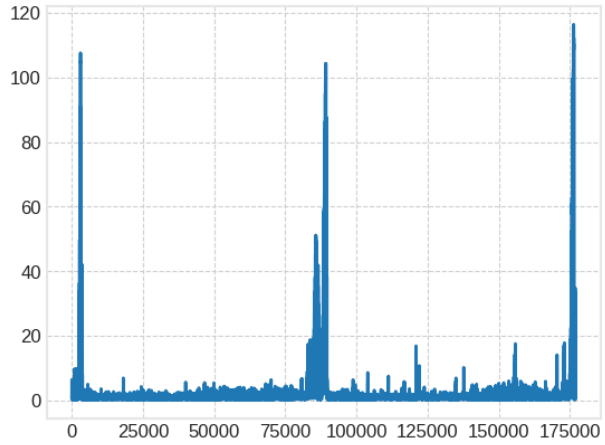
(f) Ιστόγραμμα ποσοστιαίας απόκλισης του λεπτού του scheduler

Εικόνα 1: AMZN

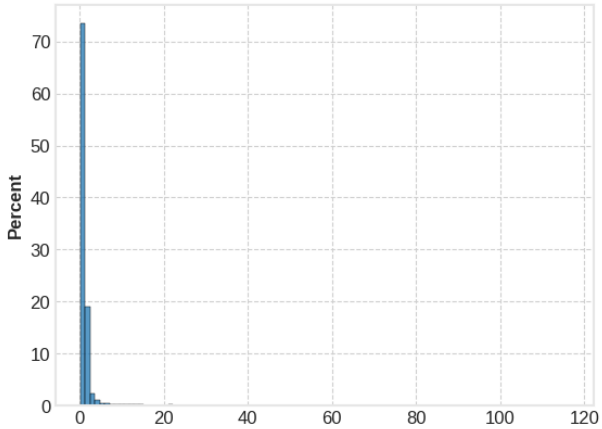
(a) Candlesticks



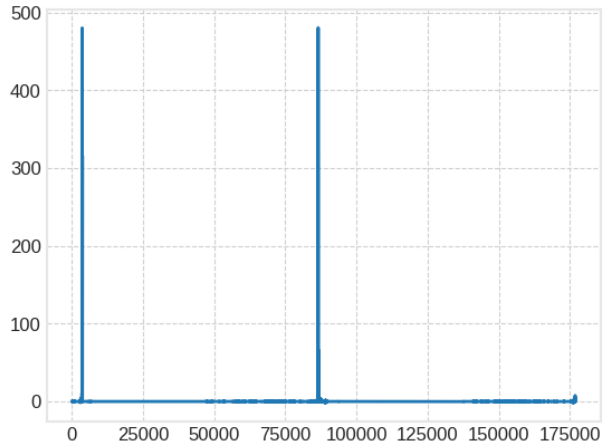
(b) Κινητός μέσος όρος τιμής 15 πιο πρόσφατων λεπτών



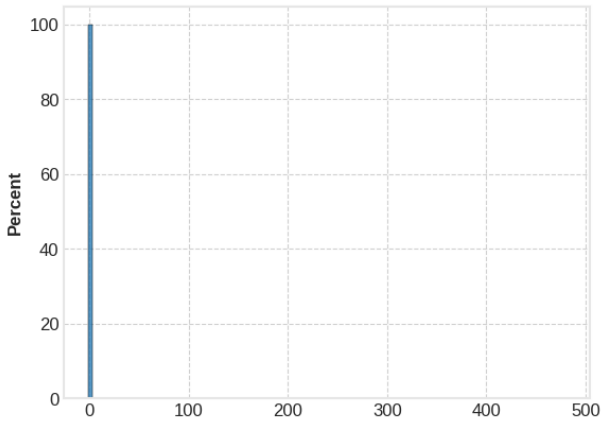
(c) Χρονική καθυστέρηση καταγραφής συναλλαγών (s)



(d) Ιστόγραμμα χρονικής καθυστέρησης καταγραφής συναλλαγών (s)



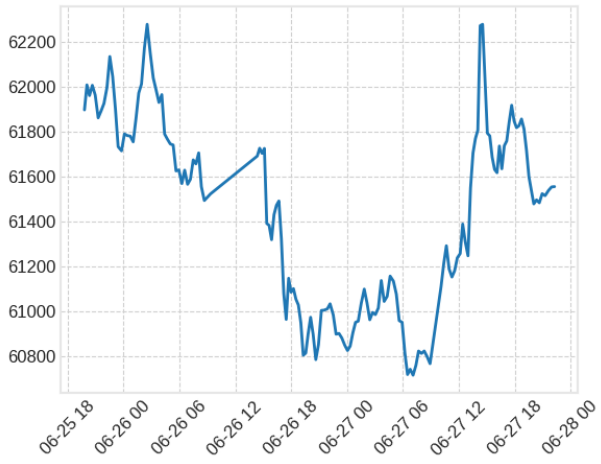
(e) Ποσοστιαία απόκλιση του λεπτού του scheduler



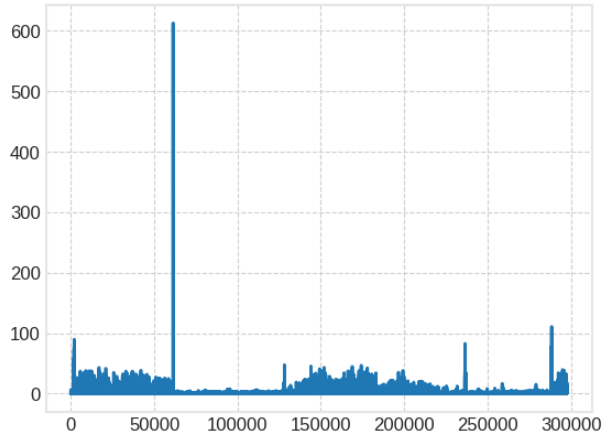
(f) Ιστόγραμμα ποσοστιαίας απόκλισης του λεπτού του scheduler

Εικόνα 2: MSFT

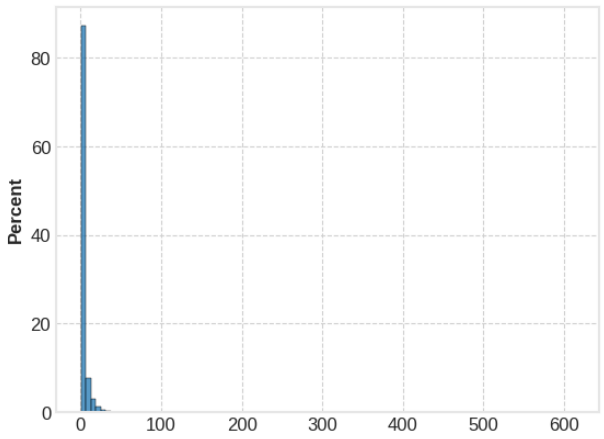
(a) Candlesticks



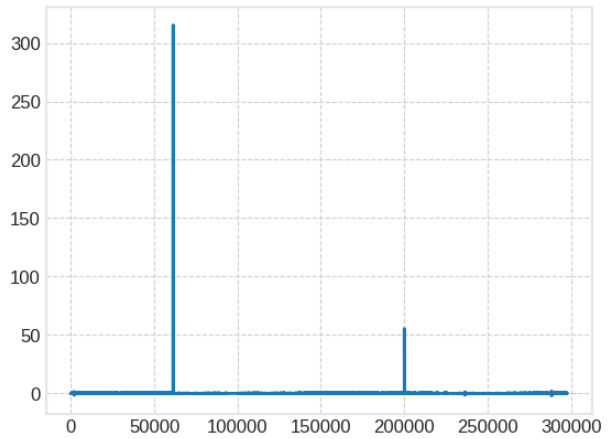
(b) Κινητός μέσος όρος τιμής 15 πιο πρόσφατων λεπτών



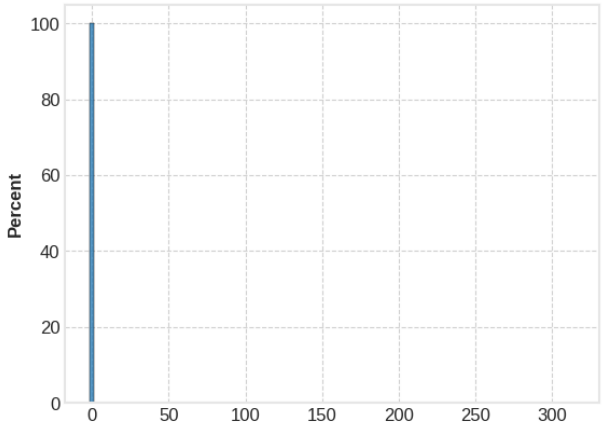
(c) Χρονική καθυστέρηση καταγραφής συναλλαγών (s)



(d) Ιστόγραμμα χρονικής καθυστέρησης καταγραφής συναλλαγών (s)



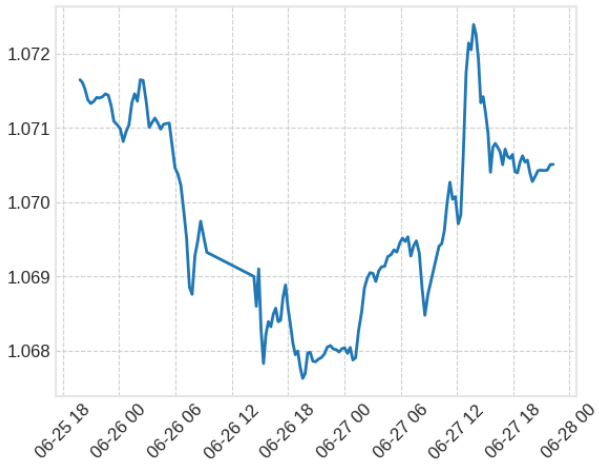
(e) Ποσοστιαία απόκλιση του λεπτού του scheduler



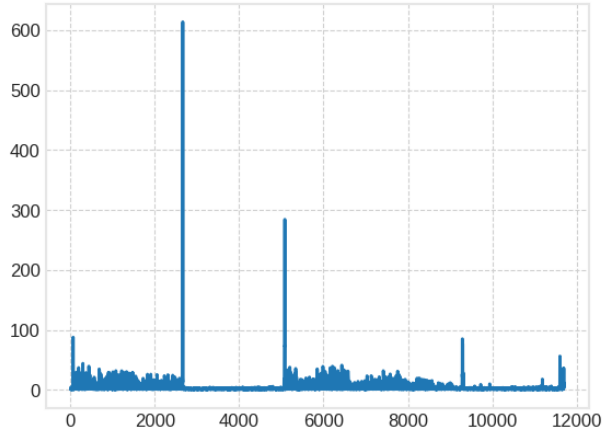
(f) Ιστόγραμμα ποσοστιαίας απόκλισης του λεπτού του scheduler

Εικόνα 3: Binance

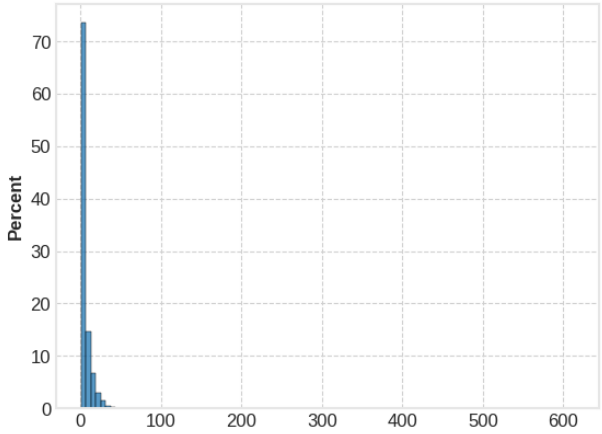
(a) Candlesticks



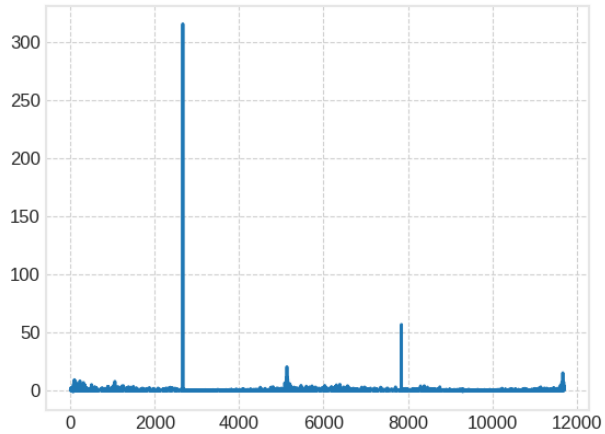
(b) Κινητός μέσος όρος τιμής 15 πιο πρόσφατων λεπτών



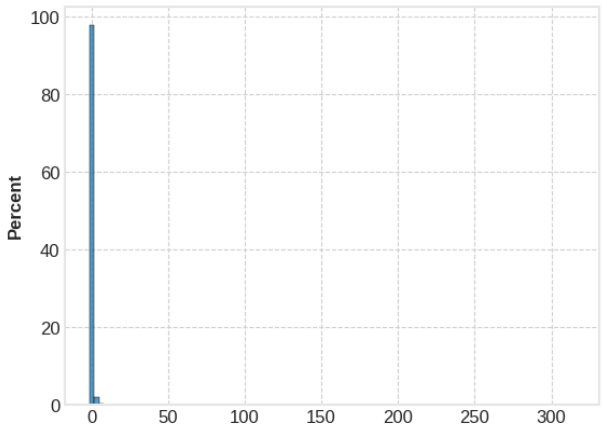
(c) Χρονική καθυστέρηση καταγραφής συναλλαγών (s)



(d) Ιστόγραμμα χρονικής καθυστέρησης καταγραφής συναλλαγών (s)



(e) Ποσοστιαία απόκλιση του λεπτού του scheduler



(f) Ιστόγραμμα ποσοστιαίας απόκλισης του λεπτού του scheduler

Εικόνα 4: IC MARKETS: 1

Για να καταγράψουμε πόσο χρόνο η CPU μένει αδρανής, χρησιμοποιήσαμε την εντολή `time`, με το ακόλουθο αποτέλεσμα:

```
./bin/client 132.68s user 38.90s system 0% cpu 50:46:18.77 total
```

Συμπεράσματα

- Υπήρχαν κάποια χρονικά διαστήματα που δεν λαμβάναμε πληροφορία για συναλλαγές. Αυτό οφείλεται στην λειτουργία της πλατφόρμας finnhub, και όχι στην δική μας υλοποίηση. Εκεί παρατηρούμε κενά στα διαγράμματα candlesticks.
- Στις χρονικές στιγμές που χανόταν η σύνδεση, και ένας από τους producers προσπαθούσε για επανασύνδεση, παρατηρούμε μεγάλες χρονικές καθυστερήσεις στα διαγράμματα χρόνου. Αυτό οφείλεται επειδή οι consumers και ο scheduler, περιμένουν να τελειώσουν την δουλειά τους οι producers, υπεύθυνοι για την επανασύνδεση, που ενδεχομένως να απαιτεί αρκετές χρονοβόρες προσπάθειες.
- Από το αποτέλεσμα της εντολής `time`, συμπεραίνουμε ότι το πρόβλημα είναι I/O bound και όχι CPU bound, μιας και η CPU την περισσότερη ώρα βρίσκεται σε αναμονή.