Αναφορά Εργασίας

Ενσωματωμένα Συστήματα Πραγματικού Χρόνου

Εαρινό Εξάμηνο 2024

Γεώργιος Σελιβάνωφ
selivanof@ece.auth.gr

5 Οκτωβρίου 2024

Περιεχόμενα

Σημείωση: *Έγινε η βέλτιστη δυνατή προσπάθεια να περιοριστεί το μέγεθος της αναφοράς στις ενδεδειγμένες 4 σελίδες, παρόλα αυτά το τελικό μέγεθος της είναι 6 σελίδες, ώστε τα γραφήματα και οι πίνακες να είναι μεγάλα και ευανάγνωστα. Πολλά σημεία της υλοποίησης αναφέρονται επιγραμματικά, καθώς έπρεπε επίσης να αναφερθούν - μεταξύ άλλων - προβλήματα του API, παραδοχές και υποθέσεις που έγιναν καθώς και ο τρόπος διασφάλισης της ορθής λειτουργίας του προγράμματος. Τα headers στο repo της εργασίας περιέχουν αναλυτικές πληροφορίες σε θέματα υλοποίησης.*

1 Βασική Δομή Προγράμματος

1.1 Τεχνικές Πληροφορίες

Για την υλοποίηση της εργασίας έγινε χρήση της γλώσσας C++. Χρησιμοποιήθηκαν οι βιλιοθήκες **libwebsockets**, **simdjson** και **fmt**.

Όσον αφορά το cross-compiling, αναλυτική περιγραφή για τα βήματα που ακολουθήθηκαν υπάρχει στο repo της εργασίας.

1.2 Σημαντικές Παρατηρήσεις

Πριν αναλυθούν ορισμένες μέθοδοι που χρησιμοποιήθηκαν, ακολουθούν συνοπτικά ορισμένες παρατηρήσεις σχετικά με την λειτουργία της εφαρμογής:

- Το API του Finnhub **δεν** στέλνει τα trades με αυστηρή χρονολιγική σειρά. Για αυτόν τον λόγο, στα πλαίσια της εφαρμογής θεωρήθηκε ότι τα trades που λαμβάνονται μπορεί να αφορούν **οποιαδήποτε** χρονική στιγμή.
- Προφανώς δεν γίνεται να ξέρουμε άμα έχουμε λάβει όλα τα μηνύματα που αφορούν το λεπτό που μόλις πέρασε, τόσο λόγο του παραπάνω όσο και των καθυστερήσεων του δικτύου. Τα ζητούμενα στατιστικά λεπτού υπολογίζονται με τα δεδομένα που έχουν ληφθεί μέχρι την χρονική στιγμή που τα γράφουμε στο αρχείο.
- Μηνμύματα που αφορούν σύμβολα **Forex** έχουν μηδενικό volume . Για τον υπολογισμό των στατιστικών, θεωρούνται όλα τα μηνύματα ισάξια με μοναδιαίο volume.
- Σε περίπτωση που στο τέλος του λεπτού ληφθούν trades με το ίδιο timestamp, closing price θεωρείται η τιμή αυτού που έπεται, θεωρώντας ότι το Finnhub έχει μεριμνήσει και τα τοποθετεί αναλόγως στο μήνυμα.

1.3 Ροή των Trade

Το κυρίως thread δέχεται και ερμηνεύει τα μηνύματα του Finnhub, αποτελώντας τον μοναδικό producer του προγράμματος. Από την πλευρά των consumer, εντοπίζουμε 2 διαφορετικά - ως προς την λειτουργία που πραγματοποιούν - είδη από workers πραγματικού χρόνου. Τους Recording Workers που καταγράφουν τα trades σε αρχεία και τους Statistics Workers που ενημερώνουν τα στατιστικά (κινούμενο μέσο όρο και candlestick).

Κάθε worker έχει το δικό του queue σύμφωνα με το μοντέλο Single Producer - Single Consumer (SP-SC) και είναι υπεύθυνο για συγκεκριμένα σύμβολα, τα οποία μοιράζονται ισάξια[1] στην αρχή του προγράμματος. Αν και η τακτική SP-SC δεν εγγυάται τον ισομερή καταμερισμό του φόρτου στα διαθέσιμα threads, πλεονεκτεί έναντι του μοντέλου Single Producer - Multiple Consumers, που θα οδηγούσε πιο συχνά σε ανταγωνισμούς για τα mutex.

[1] Το κάθε thread αναλαμβάνει ίσο αριθμό από σύμβολα. Θεωρείται ότι η λίστα συμβόλων που παρέχεται στο `websocket_config.json` περιέχει έγκυρα σύμβολα, ειδάλλως θα ληφθούν και τα άκυρα σύμβολλα υπόψιν στην κατανομή.

Μια τρίτη κατηγορία από workers αποτελούν οι Exporting Workers, οι οποίοι κάθε λεπτό (XX:XX:00) καταγράφουν τα - ήδη υπολογισμένα - στατιστικά στα κατάλληλα αρχεία. Όπως και πριν, έτσι και σε αυτήν την περίπτωση στον κάθε worker έχουν ανατεθεί συγκεκριμένα σύμβολα.

Ο αριθμός των thread που θα διατεθούν για κάθε τύπο worker είναι παραμετροποιήσημος στο runtime με την βοήθεια του αρχείου `websocket_config.json`.

2 Ανάλυση Προγράμματος

2.1 Υπολογισμός στατιστικών σε πραγματικό χρόνο

Η υπολογισμός των στατιστικών σε πραγματικό χρόνο επιτρέπει την -ανά λεπτό- εξαγωγή των στατιστικών σε αρχεία με πολύ μικρή καθυστέρηση, αφού όλα τα απαραίτητα δεδομένα έχουν ήδη υπολογιστεί σε πραγματικό χρόνο.

Η βασική δυσκολία στην υλοποίηση των μηχανισμών υπολογισμού των στατιστικών ήταν η σποραδική παραλαβή καθυστερημένων trades που αναφέρθηκε στο 1.2. Για να επιλυθεί, χρησιμοποιήθηκαν τα **hashed** `std::unordered_map` που προσφέρει η C++. Χρησιμοποιώντας timestamps της μορφής HH:MM:00 ως κλειδιά, μπορούμε σε σταθερό χρόνο O(1) να ενημερώσουμε τα στατιστικά του σωστού λεπτού για κάθε ληφθέν trade, ανεξαρτήτως της καθυστέρησης με την οποία λήφθηκαν.

Η συγκεκριμένη υλοποίηση **δεν διορθώνει** στατιστικά που έχουν ήδη καταγραφεί σε αρχεία. Για να εφαρμοστεί κάτι τέτοιο, θα ήταν προτιμότερο να χρησιμοποιηθεί βάση δεδομένων για την καταγραφή των δεδομένων, ώστε να ελαχιστοποιηθούν οι χρόνοι αναζήτησης και επεξεργασίας των ήδη αποθηκευμένων στατιστικών.

2.2 Διαχείριση αρχείων και εγγραφών

Με διαφορά η μεγαλύτερη καθυστέρηση στις εγγραφές οφειλόταν στην διαδικασία της εγγραφής σε αρχεία. Για αυτό τον λόγο δόθηκε ιδιαίτερη βαρύτητα σε αυτό το κομμάτι, ώστε να μειωθεί το overhead της εγγραφής στο ελάχιστο δυνατό. Συγκεκριμένα:

- Αντί να ανοίγουν και να κλείνουν τα αρχεία με κάθε εγγραφή ο client τα διατηρεί ανοιχτά **καθ'όλη** τη διάρκεια εκτέλεσης του προγράμματος. Αυτό προϋποθέτει ότι τα αρχεία **δεν θα διαγραφούν** από τον χρήστη στο διάστημα αυτό. Η ανάγνωση, όπως και η αντιγραφή των αρχείων δεν επηρεάζει το πρόγραμμα.
- Οι εγγραφές στα αρχεία είναι από προεπιλογή **buffered** τόσο στην C όσο και στην C++. Αυτό έρχεται σε αντίθεση με την απαίτηση της εργασίας για καταγραφή σε **πραγματικό χρόνο**. Στην εφαρμογή αυτή, χρησιμοποιήθηκαν **unbuffered** `std::ofstream` για τις εγγραφές, επιτυγχάνοντας εγγραφή σε πραγματικό χρόνο, βελτίωση της απόδοσης (σε σχέση με `fflush` μετά από κάθε εγγραφή), και μείωση της χρήσης μνήμης.

2.3 Πρακτικές ελαχιστοποίησης καθυστερήσεων

Επιγραμματικά αναφέρονται ορισμένες πρακτικές που ακολουθήθηκαν για να μειωθούν οι καθυστερήσεις στο ελάχιστο δυνατό:

- Περιορισμός των **lock** στο ελάχιστο δυνατό **scope**, μειώνοντας τον αριθμό των εν δυνάμει ανταγωνισμών για τα mutex.
- Αποφυγή αχρείαστων αντιγραφών με την χρήση **references** και της **std**::**move**.
- Ορισμός **inline** συναρτήσεων για απλές επαναλαμβανόμενες διαδικασίες.
- Πλήρης **αποφυγή** χρήσης **virtual** μεθόδων/κλάσεων.
- Χρήση **templates** έναντι εναλλακτικών, όπου ήταν εφικτό.
- Δυνατότητα **δέσμευσης των threads** σε συγκεκριμένους πυρήνες της CPU, ανάλογα την την δουλεία που εκτελούν (μέσω του config).

3 Μετρήσεις

Η πραγματοποίηση των μετρήσεων έγινε με τον μέγιστο αριθμό συμβόλων που επιτρέπει το δωρέαν API του Finnhub (50). Αποτελούνται από τα 30 πιο ενεργά σύμβολα, 10 πιο ενεργά κρυπτονομίσματα και 10 πιο ενεργά συναλλάγματα[2]. Χρησιμοποιήθηκε το `websocket_config.json` που βρίσκεται στο repo της εργασίας (εγγραφή σε δημοφιλή σύμβολα).

Αν και αρχικά οι μετρήσεις έγιναν σε ημέρες που λειτουργεί το χρηματιστήριο, παρατηρήθηκε πως είχε παραληφθεί η καταγραφή του όγκου στα στατιστικά. Αναγκαστικά, οι τελικές μετρήσεις έγιναν στις μόνες διαθέσιμες ημέρες (Παρασκευή - Σάββατο). Αν και τα 20 από τα 50 σύμβολα λειτουργούν και το Σάββατο, θεωρήθηκε σκόπιμο να χωριστούν όλες οι καθυστερήσεις σε 1ο και 2ο εικοσιτετράωρο, όπου το σύστημα λάμβανε -κατά κύριο λόγο - μηνύματα για 50 και 20 σύμβολα αντίστοιχα. Παρόλα αυτά, όπως φαίνεται και παρακάτω, δεν υπήρχαν διαφορές ανάμεσά τους.

3.1 Καθυστερήσεις καταγραφών πραγματικού χρόνου

Οι καθυστερήσεις για την εγγραφή σε πραγματικό χρόνο υπολογίζονται από την διαφορά το timestamp άφιξης των trades στο σύστημα και του timestamp ακριβώς πριν την εγγραφή[3]. Για να βεβαιωθούμε ότι το timestamp άφιξης είναι όσο το δυνατόν πιο ακριβές:

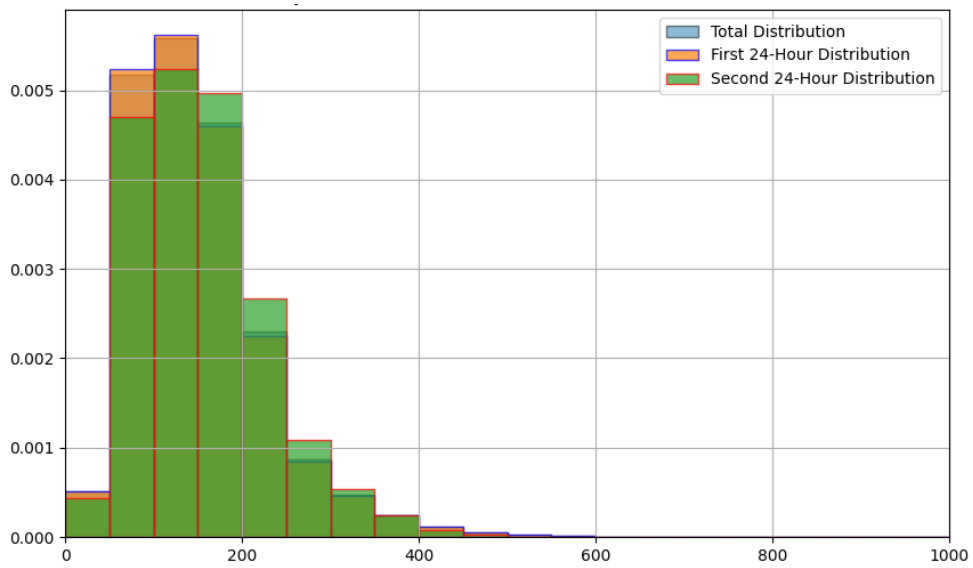
- Το timestamp άφιξης υπολογίζεται όσο το δυνατόν γρηγορότερα (πρώτη εντολή της callback άφιξης μηνύματος).
- Επιβεβαιώθηκε πως η μέγιστη διάρκεια της callback άφιξης μηνύματος (avg: 104μs, **max**: **598μs**) ήταν μικρότερη από το ελάχιστο μεσοδιάστημα λήψης μηνυμάτων από το websocket (avg: 73ms, **min**: **3ms**).
- Δεσμεύτηκε ο Πυρήνας 0 της CPU για το κυρίως thread και την αποδοχή μηνυμάτων από το Finnhub.

[2]Πηγή: Yahoo Finance

[3]Αν και το δεύτερο timestamp δεν περιλαμβάνει τον χρόνο εκτέλεσης της εντολής `write`, αυτός αποτυπώνεται **εμμέσως** αφού καθυστερείται η καταγραφή των υπόλοιπων trade που βρίσκονται στην ουρά. Το παραπάνω επιβεβαιώθηκε καταγράφοντας σε ξεχωριστό 24ωρο τη μέση τιμή των καθυστερήσεων με το δεύτερο timestamp **μετά** την εντολή εγγραφής, αποθηκεύοντας την στη μνήμη μέχρι την έξοδο του προγράμματος.

Στατιστικό	1ο 24ωρο	2ο 24ωρο	Όλο το 48ωρο
Μέση Τιμή	159.59 μs	154.7 μs	159.26 μs
Διάμεσος	137 μs	146 μs	138 μs
1% High	409 μs	376 μs	405 μs
Μέγιστο	1590652 μs	4838 μs	1590652 μs
Ελάχιστο	7 μs	10 μs	7 μs

Πίνακας 1: Στατιστικά Καθυστερήσεων εγγραφών πραγματικού χρόνου



Σχήμα 1: Κατανομές των καθυστερήσεων

Η κατανομή των καθυστερήσεων προσεγγίζει την Log-Normal[4], όπως ήταν αναμενόμενο καθώς οι παράγοντες που συμβάλλουν στην μεταβλητότητα των καθυστερήσεων (επιπλέον φόρτος συστήματος, χρόνος πρόσβασης SD κάρτας, κλπ) ακολουθούν Κανονική ή Log-Normal κατανομή. Η επικράτηση της Log-Normal έναντι της Κανονικής κατανομής οφείλεται στην ύπαρξη ενός ελάχιστου χρόνου κάτω από τον οποίο είναι πρακτικά αδύνατο να ερμηνευθεί και να καταγραφεί ένα trade.

3.2 Καθυστερήσεις καταγραφών στατιστικών ανά λεπτό

Οι καθυστερήσεις για τον υπολογισμό των στατιστικών ανά λεπτό υπολογίζονται από την διαφόρα του timestamp XX:XX:00 και του timestamp ακριβώς πριν την εγγραφή. Για να έχουν νόημα οι μετρήσεις, πρέπει να επιβεβαιώσουμε ότι ο υπολογισμός των στατιστικών σε πραγματικό χρόνο δεν καθυστερεί και άρα τα στατιστικά που καταγράφει ο exporting worker περιέχουν όλα τα trades **που έχουν ληφθεί στο σύστημα** μέχρι την χρονική στιγμή XX:XX:00. Για να επιβεβαιώσουμε τα παραπάνω:

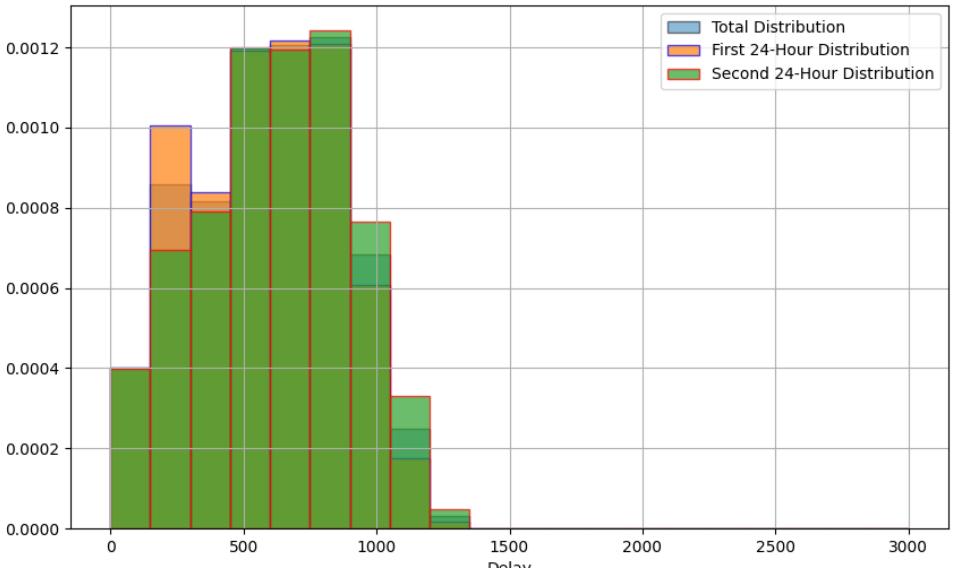
- Μετρήθηκε πως η μέση διάρκεια εκτέλεσης της `update_statistics`, που καλείται σε κάθε trade αμέσως μόλις τοποθετηθεί στο queue, είναι μόλις 1,2μs, με το υψηλότερο 1% στα 4,1μs. Δεδομένου ότι κάθε μήνυμα του finnhub έχει

[4]Q-Q plot στο `recording_delays.py`

το πολύ 10 trades, ο μέσος χρόνος υπολογισμού των στατιστικών ανά μήνυμα είναι *περίπου* 12μs.

Στατιστικό	**1ο 24ωρο**	**2ο 24ωρο**	**Όλο το 48ωρο**
Μέση Τιμή	619.1 μs	574.88 μs	595.96 μs
Διάμεσος	631 μs	587 μs	608 μs
1% High	1180 μs	1129 μs	1160 μs
Μέγιστο	1402 μs	3264 μs	3264 μs
Ελάχιστο	63 μs	71 μs	63 μs

Πίνακας 2: Στατιστικά Καθυστερήσεων εγγραφών πραγματικού χρόνου



Σχήμα 2: Κατανομές των καθυστερήσεων

3.3 Χρήση CPU

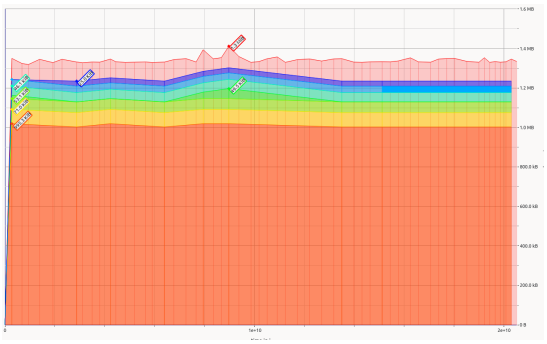
Για τον υπολογισμό του ποσοστού που η CPU έμεινε αδρανής, χρησιμοποιήθηκε η συνάρτηση `getrusage`, η οποία επιστρέφει τον συνολικό χρόνο CPU που χρησιμοποίησε η διεργασία. Το κανονικοποιημένο, ως προς τους πυρήνες του Raspberry Pi, ποσοστό που μετρήθηκε ήταν **0.15%**, χρησιμοποιώντας τον τύπο:

$$\text{Ποσοστό Χρήσης CPU} = \frac{\text{Χρόνος CPU}}{\text{Συνολικός Χρόνος Εκτέλεσης} \times \text{Φυσικοί Πυρήνες}}$$

4 Μακροχρόνια λειτουργία

4.1 Memory Profiling

Ένας πιθανός κίνδυνος για την μακροχρόνια λειτουργία του προγράμματος είναι τυχόν memory leaks που μπορεί να υπάρχουν στον κώδικα. Για αυτόν τον λόγο, πραγματοποιήθηκε ένα 24ωρο memory profiling με την χρήση Valgrind (Massif), όπου δεν παρατηρήθηκαν σχετικά προβλήματα.



Σχήμα 3: Γράφημα δέσμευσης μνήμης για 24 ώρες λειτουργίας

4.2 Διακοπές Δικτύου

Για την αντιμετώπιση τυχών διακοπών του δικτύου, χρησιμοποιήθηκε ο υπάρχων μηχανισμός αποστολής μηνυμάτων ping σε αδρανείς συνδέσεις της libwebsockets σε συνδυασμό με το αντίστοιχο timeout. Σε περίπτωση μη επιθυμητής αποσύνδεσης, το σύστημα επανασυνδέεται στο Finnhub (με χρήση εκθετικών καθυστερήσεων). Παράλληλα διατηρεί ένα connection.log με όλες τις συνδέσεις και αποσυνδέσεις συνοδευόμενες από timestamp. Όλες οι σχετικές παράμετροι δύναται να ρυθμιστούν από τον χρήστη μέσω του `websocket_config.json`.

4.3 Πράξεις αριθμών κινητής υποδιαστολής

Για τον υπολογισμό του κινούμενου μέσου όρου, καθώς και του όγκου ανά λεπτό, χρησιμοποιήθηκαν αθροίσματα αριθμών κινητής υποδιαστολής. Η αναπαράσταση τέτοιων αριθμών εμπεριέχει ένα σφάλμα το οποίο αν και μικρό, μπορεί μακροχρόνια να συσσωρευτεί και να οδηγήσει σε λανθασμένα αποτελέσματα. Για να αποφευχθεί αυτό, δεδομένου ότι το Finnhub έχει μέγιστη ακρίβεια 6 δεκαδικών ψηφίων[5], όλα τα αθροίσματα αριθμών κινητής υποδιαστολής υλοποιήθηκαν μέσω `int64_t`, πολλαπλασιάζοντας και διαιρώντας τους αριθμούς με 10^6.

[5] Αν και το API του Finnhub δεν διευκρινίζει το είδος των μεταβλητών που επιστρέφει, μετρήθηκε πειραματικά πως τα πεδία price και volume των μηνυμάτων δεν ξεπερνούσαν ποτέ τα 6 δεκαδικά ψηφία.