Ενσωματωμένα Συστήματα Πραγματικού Χρόνου

Αναφορά Εργασίας

Γεώργιος Σελιβάνωφ

18 Σεπτεμβρίου 2024

Περιεχόμενα

1 Βασική Δομή Προγράμματος

1.1 Τεχνικές Πληροφορίες

Για την υλοποίηση της εργασίας έγινε χρήση της γλώσσας C++. Αναζητήθηκαν βιβλιοθήκες υψηλής απόδοσης για να καλυφθούν οι απαιτήσεις της εφαρμογής και εν τέλει επιλέχθηκαν οι παρακάτω βιβλιοθήκες:

- **WebSocket++**: Μια αποδοτική, header only βιβλιοθήκη για WebSocket client με ελάχιστα dependencies. Επιλέχθηκε καθώς είναι μια γρήγορη και πολυδοκιμασμένη βιβλιοθήκη.
- **simdjson**: Μια εξαιρετικά γρήγορη βιβλιοθήκη για ερμήνευση JSON. Απαιτεί την υποστήριξη SIMD από τον επεξεργαστή, προϋπόθεση που πληρεί το Raspberry Pi 4. Υπάρχουν πολλές εναλλακτικές, όπως αναφορικά η RapidJSON.

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

1.2 Χρήση Κλάσεων

Καθώς η C++ χρησιμοποιεί κλάσεις για την υλοποίηση των struct, δημιουργήθηκαν ορισμένες κλάσεις με σκοπό την καλύτερη οργάνωση του κώδικα και στην διαχείριση ορισμένων πόρων. Στον κώδικα υπάρχει εκτενές documentation σε όλα τα header files για τις κλάσεις, όπου έχω προσπαθήσει να εξηγήσω αναλυτικά τις λειτουργίες τους.

1.3 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. Κατά την διάρκεια ενός λεπτού, υπάρχει πιθανότητα να ληφθούν trades που πραγματοποιήθηκαν στο προηγούμενο (και όχι μόνο) λεπτό. 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 XX:XX:00.
- 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.

1.4 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 (Single Producer - Single Consumer) 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, καθώς κάθε σύμβολο έχει διαφορετική συχνότητα πραγματοποίησης συναλλαγών, πλεονεκτεί έναντι του μοντέλου Single Producer - Multiple Consumers, που θα οδηγούσε πιο συχνά σε mutex contentions όταν διαφορετικά threads λάμβαναν trades για το ίδιο σύμβολο και αναγκαζόντουσαν να δρουν -έστω και στιγμιαία- σειριακά με τα υπόλοιπα.

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 λεπτόεξαγωγή 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 ??. Για να επιλυθεί, χρησιμοποιήθηκαν τα hashed **`std::unordered_map`** που προσφέρει η C++. Κάθε trade που λαμβάνεται σε χρονική στιγμή HH:MM:SS θεωρούμε ότι ανήκει στο λεπτό HH:MM:00, και χαρακτηρίζεται από το αντίστοιχο κάτω όριο του timestamp. Δημιουργώντας ένα unordered_map από timestamps της μορφής HH:MM:00 σε αντίστοιχα structs για το candlestick και τον μέσο όρο (του συγκεκριμένου λεπτού), μπορούμε σε σταθερό χρόνο O(1) να ενημερώσουμε το σωστό στατιστικό για το ληφθέν trade, ανεξαρτήτως της καθυστέρησης με την οποία στάλθηκε από το API.

Για την διαχείριση των στατιστικών δημιουργήθηκαν οι κλάσεις **Candlestick Manager** και **Moving Average**. Και οι δύο δέχονται trades και τα τοποθετούν στα σωστά containers ανάλογα με το timestamp τους. Οι δύο κλάσεις διαθέτουν getters για τα αντίστοιχα στατιστικά, οι οποίοι όταν κληθούν σε οποιαδήποτε χρονική στιγμή του διαστήματος HH:MM:00 έως HH:MM:59, επιστρέφουν το candlestick

[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.

για το λεπτό HH:(MM-1):00 έως και HH:MM:00 τον κινούμενο μέσο όρο για το διάστημα HH:(MM-15):00 έως HH:MM:00 αντίστοιχα. Λεπτομέρειες για την υλοποίηση τους υπάρχουν με την μορφή σχολίων στα αρχεία finnhub_data.h και finnhub_data.cpp.

2.2 Διαχείριση αρχείων/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, σπαταλώντας τεράστια ποσά χρόνου, 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. Σε αντίθετη περίπτωση απλώς θα πάψουν να καταγράφονται δεδομένα για το αντίστοιχο σύμβολο.

- 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 πραγματική εγγραφή σε πραγματικό χρόνο, βελτίωση της απόδοσης (σε σχέση με το `fflush` μετά από κάθε εγγραφή), και μείωση της χρήσης μνήμης.

- Παρατηρήθηκε ότι σημαντικό ρόλο στις καθυστερήσεις εγγραφών έπαιζε και το filesystem. Στις μετρήσεις μου, το `ext4` παρουσίαζε τυχαίες αιχμές στις καθυστερήσεις που επηρέαζαν σημαντικά την μέση τιμής τους (αλλά όχι την διάμεσο). Όλα τα άλλα filesystems που δοκιμάστηκαν δεν παρουσίαζαν αυτό το πρόβλημα (`exFAT, f2fs, xfs`).

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 contention.

- Αποφυγή αχρείαστων αντιγραφών με την χρήση references και της std::move.

- Ορισμός inline συναρτήσεων για επαλαμβανόμενες 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

Όλες οι μετρήσεις που αναφέρονται παρακάτω, είτε αφορούν ζητούμενα της εργασίας, είτε επιπλέον μετρήσεις που έγιναν (π.χ. χρόνοι εκτέλεσης συναρτήσεων) πραγματοποιήθηκαν σε ώρες και μέρες που όλα τα σύμβολα ήταν ενεργά. Η εφαρμογή εκτελέσθηκε σε Raspberry Pi 4 με Raspberry Pi OS Lite 64-bit και η αποθήκευση των `.csv` αρχείων έγινε σε xsf partition. Χρησιμοποιήθηκε το `websocket_config.json` που βρίσκεται στο repo της εργασίας. Επιγραμματικά:

- **Σύμβολα**: 50 (δημοφιλή stocks, crpyto και forex)
- **Recording Workers**: 2 (Πυρήνες 1 και 2)
- **Statistics Workers**: 1 (Πυρήνας 3)
- **Exporting Workers**: 1 (Πυρήνας 3)

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. Για να βεβαιωθούμε ότι το timestamp άφιξης είναι όσο το δυνατόν πιο ακριβές:

- Το timestamp υπολογίζεται με το που κληθεί η `on_message` και όχι την στιγμή της ερμηνείας του json.
- Επιβεβαιώθηκε πως η διάρκεια της `on_message` (avg: 104μs, max: 598μs) ήταν μικρότερη από το μεσοδιάστημα λήψης μηνυμάτων από το websocket (avg: 100ms, min: 3ms), από μετρήσεις 10 ωρών.
- Δεσμεύτηκε ο Πυρήνας 0 της CPU αποκλειστικά για το κυρίως thread και την αποδοχή μηνυμάτων από το Finnhub.

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 (με απόκλιση ±1%) καταγράφοντας σε ξεχωριστό 24ωρο τη μέση τιμή των καθυστερήσεων με το timestamp **μετά** την εγγραφή, αποθηκεύοντας τα στατιστικά στη μνήμη μέχρι την έξοδο του προγράμματος. Αυτή η μέθοδος αποτελεί την βέλτιστη - ως προς την ακρίβεια - λύση, καθώς μετράει την συνολική καθυστέρηση με ελάχιστο overhead. Παρόλα αυτά δεν επιτρέπει την ανάλυση των καθυστερήσεων μετά την έξοδο του προγράμματος και για αυτό δεν προτιμήθηκε. Να σημειωθεί ότι, η εγγραφή των στοιχείων του trade πριν τον υπολογισμό του δεύτερου timestamp και η μετέπειτα εγγραφή της καθυστέρησης με δεύτερη εντολή `write`, επέφερε **πλασματική** αύξηση της καθυστέρησης (σχεδόν διπλάσια απο την μέθοδο αποθήκευσης στην μνήμη), όπως διαπιστώθηκε από ξεχωριστή 24ωρη μέτρηση, οπότε και δεν επιλέχθηκε. Τα αποτελέσματα φαίνονται παρακάτω:

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. Ομοίως με πριν, οι χρόνοι των εντολών εγγραφής αποτυπώνονται εμμέσως. Για να έχουν νόημα οι μετρήσεις, πρέπει να επιβεβαιώσουμε ότι ο υπολογισμός των στατιστικών σε πραγματικό χρόνο δεν καθυστερεί και άρα τα δεδομένα που καταγράφει ο exporting worker περιέχουν όλα τα trades που έχουν ληφθεί στο σύστημα μέχρι την χρονική στιγμή XX:XX:00. Για να το πετύχουμε αυτό:

- 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 το πολύ 10 trades, ο μέσος χρόνος υπολογισμού των στατιστικών ανά μήνυμα είναι μόλις 12μs. Είναι ξεκάθαρο ότι τα στατιστικά ενημερώνονται εγκαίρως.

- Για επιπλέον επιβεβαίωση, δημιουργήθηκε το `validate_statistics.py` που υπολογίζει εκ νέου τα στατιστικά από τα αρχεία καταγραφών (λαμβάνοντας υπόψη ποια trades είχαν καταφθάσει στο σύστημα την χρονική στιγμή XX:XX:00) και τα συγκρίνει με αυτά της εφαρμογής. Αν και εν δυνάμει υπάρχει περίπτωση να εμφανιστεί false negative, άμα ο exporting worker έχει καθυστερήσει αρκετά ώστε να ληφθούν υπόψη στα στατιστικά trades με timestamp προηγούμενου λεπτού αλλά που έφτασαν στο σύστημα μετά το XX:XX:00, όλα τα αποτελέσματα του 48ώρου επαληθεύτηκαν επιτυχώς.

Τα αποτελέσματα φαίνονται παρακάτω:

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 ποσοστό, ως προς τους πυρήνες του Raspberry Pi, που μετρήθηκε ήταν **0,09234%**, χρησιμοποιώντας τον τύπο:

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

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. Το γράφημα της δέσμευσης μνήμης φαίνεται παρακάτω:

4.2 Διακοπές Δικτύου

Για την αντιμετώπιση τυχών διακοπών του δικτύου, υλοποιήθηκε ένας μηχανισμός ping-pong για την ανίχνευση των νεκρών συνδέσεων. Σε τέτοια περίπτωση, το σύστημα περιμένει για λίγο (με χρήση εκθετικού backoff), πριν προσπαθήσει να επανασυνδεθεί στο websocket και να συνεχίσει την καταγραφή. Παράλληλα διατηρεί ένα connection.log με όλες τις συνδέσεις και αποσυνδέσεις συνοδευόμενες από timestamp, ώστε αν είναι εφικτό να βρεθούν τα στατιστικά που δεν περιέχουν την μέγιστη δυνατή πληροφορία λόγω της αποσύνδεσης. Όλες οι σχετικές παράμετροι δύναται να ρυθμιστούν από τον χρήστη μέσω του `websocket_config.json`.