Ενσωματωμένα Συστήματα Πραγματικού Χρόνου - Report

Σπυρίδων Μπάλτσας

AEM: 10443

1 Σύνοψη

Η παρούσα αναφορά έχει θέμα το πρόγραμμα που υλοποιήθηκε στο πλαίσιο της εργασίας του μαθήματος των ενσωματωμένων συστημάτων πραγματικού χρόνου για το ακαδημαϊκό έτος 2023-2024. Πιο συγκεκριμένα, θα περιγραφεί αναλυτικά η υλοποίηση του προγράμματος και θα παρουσιαστούν τα αποτελέσματα από την χρήση του, ώστε να αποδειχθεί πως ικανοποιεί τις απαιτούμενες προϋποθέσεις μιας εφαρμογής που εκτελείται σε ένα ενσωματωμένο σύστημα για οσοδήποτε μεγάλο χρονικό διάστημα.

1.1 Πηγαίος κώδικας, compilation και χρήση

Ο πηγαίος κώδικας είναι διαθέσιμος στο GitHub. Για οδηγίες σχετικά με το compilation και την χρήση του μπορούν να βρεθούν στο σχετικό README που δημιουργήθηκε για αυτό το σκοπό.

2 Περιγραφή του προγράμματος

2.1 Βιβλιοθήκες

Το πρόγραμμα χρησιμοποεί την βιβλιοθήκη *libwebsockets* για την απαιτούμενη συνδεσιμότητα μέσω του πρωτοκόλλου websocket με το FinnHub. Επίσης, χρησιμοποιεί την *pthreads* για την δημιουργία νημάτων για την επεξεργασία σε πραγματικό χρόνο. Τέλος, για την εύκολη, γρήγορη και αξιόπιστη επεξεργασία των απαντήσεων του server οι οποίες είναι σε μορφή JSON[1] χρησιμοποιήθηκε η βιβλιοθήκη *cJSON*.

2.2 Υλοποίηση

Το πρόγραμμα διαβάζει ένα αρχείο που περιέχει τα σύμβολα του ενδιαφέροντας μας, όπως το *symbols.txt* και το κλειδί του API του FinnHub που απαιτείται για την σύνδεση με τον server. Έπειτα ταξινομεί τα σύμβολα αλφαβητικά μέσω του αλγορίθμου QuickSort[2] ώστε να μπορούμε έπειτα να τα αναζητάμε εύκολα μέσω της χρήσης Binary Search που εγγυάται χαμηλή πολυπλοκότητα ($\mathcal{O}(logn)$)[3]. Η αναζήτηση αυτή συμβαίνει πολύ συχνά, διότι απαιτείται κατά την λήψη των δεδομένων ώστε να γίνεται η αντιστοίχηση των συναλλαγών στα στατιστικά του συμβόλου στο οποίο ανήκουν.

Έπειτα, δημιουργούνται συνολικά 5 νήματα αναλαμβάνουν το καθένα από μία αρμοδιότητα:

- Λήψη δεδομένων από τον server, μέσω websockets, και διαχείρηση της σύνδεσης
- Υπολογισμός candlestick
- Υπολογισμός κινούμενου απλού μέσου όρου (SMA) των τελευταίων 15 λεπτών
- Αποθήκευση των στατιστικών σε αρχείο κάθε 1 λεπτό, όταν αυτό απαιτείται

- Αποθήκευση όλων των συναλλαγών σε αρχεία, μόλις ληφθούν

Οι εισερχόμενες συναλλαγές διοχετεύονται από το νήμα που λαμβάνει τα δεδομένα στα νήματα που είναι υπεύθυνα για την επεξεργασία και την απευθείας αποθήκευση με την χρήση διανυσμάτων των οποίων το μέγεθος είναι δυναμικό (struct `Vector`). Για την εξάλειψη της πιθανότητας race condition στα διανύσματα, χρησιμοποιήθηκε mutex. Επίσης, για τον υπολογισμό του κινούμενου μέσου όρου χρησιμοποιήθηκε ουρά FIFO πεπερασμένου μήκους (struct `Queue`). Στην περίπτωση μας έχει μέγεθος ίσο με το μέγιστο πλήθος των λεπτών που συμμετάσχουν στον υπολογισμό, δηλαδή 15. Ακόμη, για τον υπολογισμούς δημιουργούνται πίνακες από struct `Candle` και `MovingAverage` με πλήθος στοιχείων ίσο με το αριθμό των συμβόλων.

Να σημειωθεί ότι μερικές φορές το FinnHub έχει παρατηρηθεί πως αποστέλλει δεδομένα παλιότερων λεπτών και όχι του τρεχούμενου. Τα δεδομένα που αναφέρονται σε παλιότερα λεπτά επιλέχθηκε στην παρούσα υλοποίηση να αγνοηθούν.

2.2.1 Δομές δεδομένων

Δημιουργήθηκαν τα ακόλουθα structs. Το διάνυσμα και η ουρά δημιουργήθηκαν έτσι, ώστε να μπορούν να επαναχρησιμοποιηθούν για οποιοδήποτε τύπο δεδομένων:

```
typedef struct {
    void* data;
    size_t size;
    size_t capacity;
    size_t elemSize;
    pthread_cond_t* isEmpty;
    pthread_mutex_t* mutex;
} Vector;
typedef struct {
      void *data;
      size_t elemSize, head, tail,size,capacity;
      bool isFull, isEmpty;
  } Queue;
 typedef struct {
        size_t symbolID;
        time_t timestamp;
        double price, volume;
        struct timespec insertionTime;
 } Trade;
 typedef struct {
        Trade first,last,max,min;
        double totalVolume;
        size_t symbolID;
 } Candle;
 typedef struct {
        Trade first;
        double totalVolume;
        double averagePrice;
        size_t symbolID,tradeCount;
        time_t stopTime;
 } MovingAverage;
```

2.2.2 Υπολογισμός candlestick

Όταν υπάρχουν συναλλαγές στο διάνυσμά του, τότε υπολογίζεται σε πραγματικό χρόνο η μέγιστη και ελάχιστη τιμή του συμβόλου του τελευταίου λεπτού. Όταν περάσει το λεπτό και έρχεται η πρώτη συναλαγή από το επόμενο λεπτό, τότε το τελικό αποτέλεσμα αποθηκεύεται σε ένα struct `Candle` και ο υπολογισμός επαναρχικοποιείται.

2.2.3 Υπολογισμός κινούμενου μέσου όρου

Όταν υπάρχουν συναλλαγές στο διάνυσμά του, τότε υπολογίζεται σε πραγματικό χρόνο ο μέσος όρος του τελευταίου λεπτού και προστίθεται στην ουρά. Αν υπάρχουν δεδομένα στην ουρά παλιότερα απο τα τελευταία 15 λεπτά, διαγράφονται από την μνήμη και αφαιρούνται απο τον τελικό μέσο όρο. Επομένως, όταν έρθει η πρώτη συναλλαγή από το επόμενο λεπτό, αν η ουρά έχει γεμίσει επειδή έχουμε συμπληρώσει τα τελευταία 15 λεπτά ή να μην έχει γεμίσει έχουμε διαγράψει δεδομένα από αυτή, το οποίο συμβαίνει όταν για παράδειγμα δεν έχουμε συναλλαγές για κάποια λεπτά. Τότε υπολογίζεται και αποθηκεύεται ο τελικός μέσος όρος σε ένα struct `MovingAverage` και ο υπολογισμός επαναρχικοποιείται.

Σε περίπτωση που έχουμε μπει στο επόμενο λεπτό, δεν έχουν έρθει συναλλαγές ακόμα του επόμενου λεπτού, και το νήμα που αποθηκεύει τα στατιστικά πρέπει να ξεκινήσει να ανανεώνει τα αρχεία, τότε χρησιμοποιούνται τα τελευταία στατιστικά, όταν είναι αυτό δυνατό. Για να μειωθεί η πιθανότητα να συμβεί αυτό και να έχουμε ακριβέστερη και πληρέστερη πληροφορία, το νήμα δίνει μια ”περίοδο χάριτος” 15 δευτερολέπτων πριν ξεκινήσει την ανανέωση των αρχείων.

3 Αποτελέσματα

3.1 Τεχνικές πληροφορίες

Το πρόγραμμα εκτελέστηκε σε ένα Raspberry Pi 1 model B με τα εξής τεχνικά χαρακτηριστικά:

- CPU: ARM1176JZFS 1 core @ 700 MHz
- Αρχιτεκτονική: ARMV6
- RAM: 512 MB

Για την αποθήκευση των δεδομένων χρησιμοποιήθηκε μια Class 10 64GB SD card. Επομένως, όπως είναι ήδη φανερό, τα πιθανά bottlenecks στην επίδοση πέραν της υλοποίησης φυσικά, είναι η CPU και η ταχύτητα εγγραφής της κάρτας SD (random write). Η βιβλιοθήκη libwebsockets είναι statically linked με το υπόλοιπο πρόγραμμα. Για τον υπολογισμό του χρόνου αδράνειας της CPU και του χρόνου χρήσης της CPU από το πρόγραμμα χρησιμοποιήθηκε το εργαλείο *perf* [4] τρέχοντας την εξής εντολή ταυτόχρονα, στο παρασκήνιο:

```
perf stat -p <pid> -e task-clock,cache-references,cache-misses,context-switches,branch-misses,branches
↪  -o perf.txt
```

Συλλέχθηκαν πρόσθετα δεδομένα συστήματος, όπως συνολικό ποσοστό χρήσης CPU από το user space, χρήση της RAM και άλλα, μέσω του προγράμματος *Telegraf* και αποθηκεύτηκαν σε βάση δεδομένων *InfluxDB*, μια βάση δεδομένων χρονοσειρών, η οποία ήταν διαθέσιμη στο τοπικό δίκτυο.

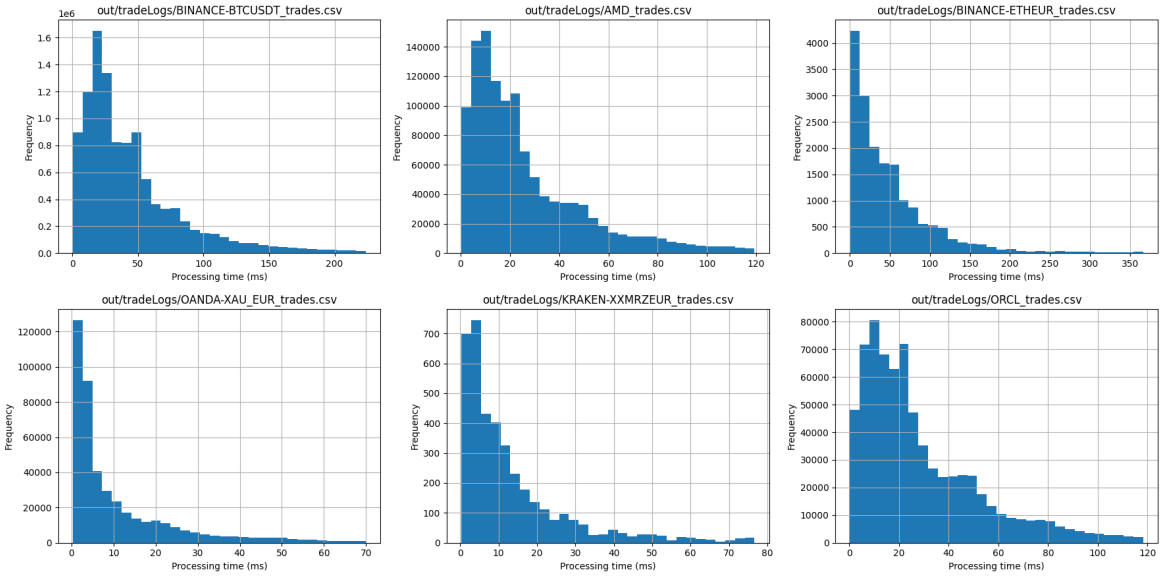
Τέλος, τα σύμβολα που επιλέχθηκαν είναι *BINANCE:BTCUSDT*, *ORCL*, *AMD*, *BINANCE:ETHEUR*, *OANDA:XAU_EUR* και *KRAKEN:XXMRZEUR*. Επιλέχθηκαν έτσι ώστε να διαφέρουν τόσο στην συχνότητα όσο και την ποσότητα των συναλλαγών, καθώς και για να καλυφθούν όλα τα είδη των συναλλαγών (μετοχές, κρυπτονομίσματα, συναλλάγματα, πολύτιμα αγαθά).

3.2 Αδράνεια CPU

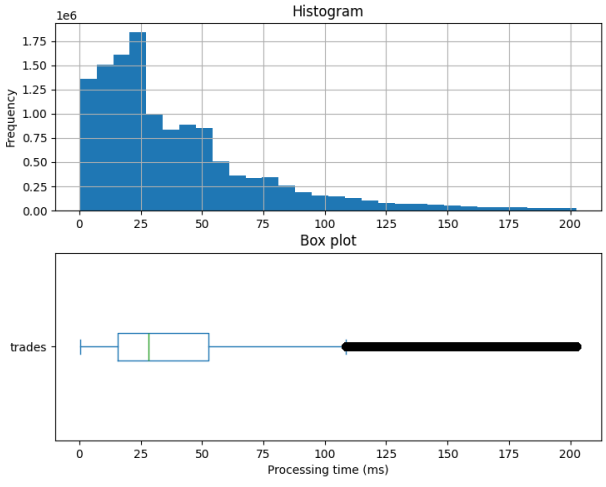
Σύμφωνα με την έξοδο του *perf* (βλέπε το αρχείο perf.txt) ο χρόνος που εκτελέστηκε το πρόγραμμα ήταν περίπου 273996 δευτερόλεπτα και η διάρκεια που χρησιμοποίησε την CPU ήταν περίπου 12875 δευτερόλεπτα. Επομένως το ποσοστό της αδράνειας της CPU είναι 96%, το οποίο είναι ένα καλό ποσοστό για εφαρμογή που τρέχει σε ενσωματωμένο σύστημα, διότι όσο μεγαλύτερο είναι το ποσοστό αδράνειας του επεξεργαστή, τόση λιγότερη ενέργεια καταναλώνεται[5].

3.3 Διαγράμματα

3.3.1 Κατανομή χρόνου επεξεργασίας συναλλαγών

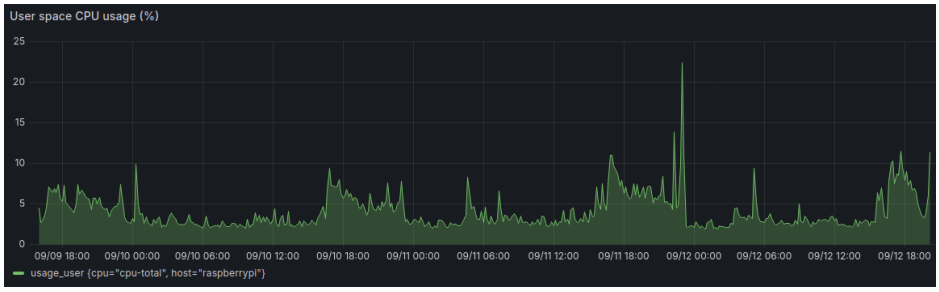


Σχήμα 1: Ιστογράμματα χρόνου επεξεργασίας ανά σύμβολο

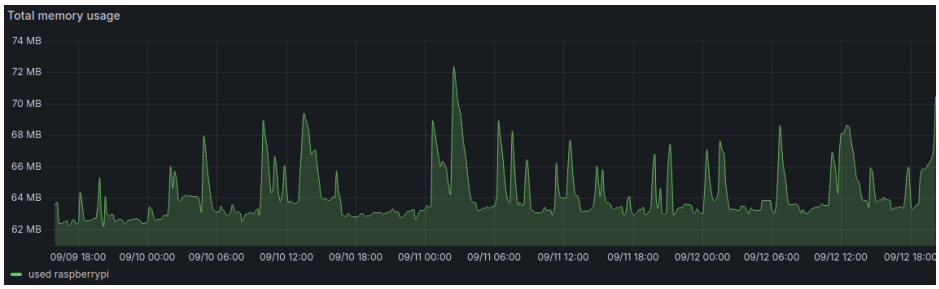


Σχήμα 2: Ιστογράμμα και θηκόγραμμα του χρόνου επεξεργασίας όλων των συμβόλων

3.3.2 Πόροι συστήματος



Σχήμα 3: Συνολικό ποσοστό χρήσης του επεξεργαστή από το user space κατά την εκτέλεση



Σχήμα 4: Συνολική χρήση μνήμης κατά την εκτέλεση

3.4 Παρατηρήσεις

Σύμφωνα με τα διαγράμματα 1 και 2, ο χρόνος επεξεργασίας των συναλλαγών ακολουθεί γεωμετρική κατανομή, όπως ήταν αναμενόμενο. Επιπλέον, από το θηκόγραμμα του σχήματος 2, φαίνεται πως έχουμε πολλά πιθανά outliers, το οποίο εξηγείται από την συμπεριφορά του τρόπου αποθήκευσης των δεδομένων από το λειτουργικό σύστημα (ΛΣ), αλλά και απο την φύση της κατανομής. Για παράδειγμα, η αποθήκευση των δεδομένων μπορεί να καθυστερήσει πολύ όταν το ΛΣ δίνει προτεραιότητα σε εγγραφές άλλων προγραμμάτων. Ακόμη, η καθυστέρηση αυτή εξηγείται από την ταχύτητα αποθήκευσης των δεδομένων στην κάρτα SD ή αν την δεδομένη χρονική στιγμή που θέλουμε να εγγράψουμε η κάρτα είναι απασχολημένη για οποιοδήποτε άλλον, άγνωστο για εμάς, λόγο. Τέλος, η μέση καθυστέρηση θα ήταν αρκετά χαμηλότερη τα δεδομένα αποθηκευόταν σε έναν σκληρό δίσκο, είτε μηχανικό είτε SSD.

4 Αναφορές

[1] FinnHub, *FinnHub API documentation*, `https://finnhub.io/docs/api/websocket-trades`, [Online; accessed 16-September-2024], 2024.

[2] Wikipedia contributors, *Quicksort — Wikipedia, The Free Encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Quicksort&oldid=1241885213`, [Online; accessed 15-September-2024], 2024.

[3] Wikipedia contributors, *Binary search — Wikipedia, The Free Encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Binary_search&oldid=1242941867`, [Online; accessed 15-September-2024], 2024.

[4] Linux Foundation, *perf: Linux profiling with performance counters*, `https://perf.wiki.kernel.org/index.php/Main_Page`, [Online; accessed 15-September-2024], 2024.

[5] S. Daud, R. Ahmad, B. Ong κ.ά., "The Effects of CPU Load Idle State on Embedded Processor Energy Usage", Αύγ. 2014. DOI: `10.1109/ICED.2014.7015766`.