Ατομική Διπλωματική Εργασία

ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΟΥ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΚΑΤΑΣΤΙΧΩΝ ΑΝΘΕΚΤΙΚΟ ΣΕ ΒΥΖΑΝΤΙΝΑ ΣΦΑΛΜΑΤΑ ΜΕ ΤΟ ΕΡΓΑΛΕΙΟ BFT-SMART

Ανδρέας Χρυσάνθου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΟΥ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΚΑΤΑΣΤΙΧΩΝ ΑΝΘΕΚΤΙΚΟ ΣΕ ΒΥΖΑΝΤΙΝΑ ΣΦΑΛΜΑΤΑ ΜΕ ΤΟ ΕΡΓΑΛΕΙΟ BFT-SMART

Ανδρέας Χρυσάνθου

Επιβλέπων Καθηγητής
Χρύσης Γεωργίου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2022

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου κ. Χρύση Γεωργίου, ο οποίος μέσα από τις συναντήσεις μας με καθοδηγούσε ώστε να ολοκληρώσω την διπλωματική μου εργασία, με ενθάρρυνε και μου κάλυπτε όλες μου τις απορίες τόσο σε προγραμματιστικό όσο και σε θεωρητικό επίπεδο. Μέσα από την συνεργασία μου με τον κύριο Χρύση Γεωργίου κατάλαβα την σημαντικότητα των κατανεμημένων αλγορίθμων και της συστηματικής αναβάθμισης των γνώσεων μας.

Επίσης θα ήθελα να ευχαριστήσω θερμά τους κυρίους Robin Vassantlal και Alysson Bessani οι οποίοι είναι μέλη της ομάδας που δημιούργησαν το εργαλείο BFT-Smart, και συνέβαλαν στην ολοκλήρωση της ΑΔΕ μέσα από την επίλυση απορίων σχετικά με το BFT-Smart.

Μέσα από την ΑΔΕ είχα την δυνατότητα να βελτιωθώ σε διάφορους τομείς της πληροφορικής, όπως η επικοινωνία μεταξύ υπολογιστών και η δημιουργία γραφικής διεπαφής, η οποία κάνει την αλληλεπίδραση μεταξύ χρηστών και συστήματος πιο εύχρηστη. Τέλος η ενασχόληση με την πτυχιακή βοήθησε στο να κατανοήσω την σημαντικότητα την συνεχής μελέτης αφού η γνώση δεν τελειώνει ποτέ.

Περίληψη

Ζούμε σε μια εποχή η οποία χαρακτηρίζεται από την άνθιση των κρυπτονομισμάτων και των τεχνολογιών που τα υποστηρίζουν. Αυτή η ανάπτυξη προκύπτει από την απελευθέρωση των κατόχων κρυπτονομισμάτων από ασφαλή κεντρικοποιημένες αρχές (τράπεζες), και στην αύξηση του ενδιαφέροντος για αποκεντρωμένα συστήματα συναλλαγών (σε ένα τέτοιο σύστημα συναλλαγών βασίζεται και το Bitcoin) τα οποία βασίζονται σε τεχνολογίες όπως τα Κατανεμημένα Κατάστιχα (Distributed Ledgers) και οι Αλυσίδες Κοινοποίησης (Blockchains).

Οι Αλυσίδες Κοινοποίησης προσφέρουν αξιοπιστία μιας και αποθηκεύουν τα δεδομένα σε διάφορους εξυπηρετητές, αυτό όμως δημιουργεί διάφορες προκλήσεις όπως η διατήρηση της συνέπειας των δεδομένων των εξυπηρετητών.

Στόχος της διπλωματικής μου εργασίας είναι η υλοποίηση και ανάπτυξη γραφικής διεπαφής για ένα αλγόριθμο Κατανεμημένων Αντικειμένων Κατάστιχων (ΚΑΚ) ο οποίος είναι ανθεκτικός σε Βυζαντινά σφάλματα. Ο αλγόριθμος βασίζεται πάνω σε μια υπηρεσία ατομικής πολυεκπομπής ανθεκτικής σε Βυζαντινά σφάλματα (Byzantine-tolerant Atomic Broadcast ~ BAB) και για αυτή την υπηρεσία έγινε η χρήση του BFT-Smart, μιας βιβλιοθήκης για τη γλώσσα Java (η γλώσσα δηλαδή όπου υλοποιήθηκε και το σύστημα).

Οι βασικές οντότητες είναι οι πελάτες και οι εξυπηρετητές. Το κατάστιχο (Ledger) είναι κατανεμημένο στους εξυπηρετητές (ο κάθε εξυπηρετητής έχει το δικό του αντίγραφο) και περιέχει εγγραφές. Οι πελάτες μπορούν να προσθέσουν εγγραφές με την λειτουργία πρόσθεση (append) και να λάβουν το περιεχόμενο του κατάστιχου με την λειτουργία λήψη (get). Οι εξυπηρετητές ανάλογα προσθέτουν εγγραφές ή επιστρέφουν το περιεχόμενο τους. Στην διπλωματική υλοποιήθηκαν οι πιο πάνω οντότητες καθώς και ένας διαχειριστής ο οποίος θα μπορεί να βλέπει ανά πάσα στιγμή τα περιεχόμενα του αντικειμένου κατάστιχου κάθε εξυπηρετητή.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο

Τα τελευταία χρόνια υπάρχει ένα έντονο ενδιαφέρον σε τεχνολογίες οι οποίες είναι γνωστές με το όνομα κρυπτό-τεχνολογίες (όπως τα Συστήματα Αλυσίδων Κοινοποίησης ~ Blockchain Systems) και γενικά σε κατανεμημένα κατάστιχα (Distributed Ledgers).

Οι πιο πάνω τεχνολογίες γίνονται όλο και πιο δημοφιλή, έτσι αναμένεται να έχουν μεγάλη επίδραση σε πολλούς τομείς της ζωής μας. Ο λόγος αυτής της άνθισης πέρα από την τρομερή ανάπτυξη των κρυπτονομισμάτων (όπου βασίζονται σε τέτοιου είδους τεχνολογίες), είναι τα χαρακτηριστικά τους τα οποία παρέχουν ασφάλεια, αξιοπιστία και διαφάνεια. Τέτοια χαρακτηριστικά είναι η αποκεντρωμένη (decentralized) διατήρηση πληροφοριών και η αμετάβλητη τήρηση εγγραφών σε ένα σύνολο από διαφορετικούς εξυπηρετητές.

Τα πιο πάνω αποτέλεσαν κίνητρο για την δημιουργία ενός συστήματος που υλοποιεί την γραφική διεπαφή για τον αλγόριθμο Κατανεμημένων Αντικειμνων Κατάστιχου (ΚΑΚ) ανθεκτικό σε Βυζαντινά σφάλματα.

1.2 Στόχος Διπλωματικής Εργασίας

Στόχος της Διπλωματικής Εργασίας είναι η μελέτη, κατανόηση και υλοποίηση του αλγορίθμου. Η δημιουργία μιας διεπαφής για τους πελάτες, τους εξυπηρετητές και για τον διαχειριστή του συστήματος, οπού οι χρήστες (μέσω πελατών) μπορούν με ευκολία να επικοινωνούν με τους εξυπηρετητές και να πραγματοποιούν αιτήματα πρόσθεσης και λήψης. Οι εξυπηρετητές να παρουσιάζουν τις πιο πρόσφατες εγγραφές του αντίγραφου κατάστιχου τους και ο διαχειριστής να παρουσιάζει τις εγγραφές κατάστιχου για όλους τους εξυπηρετητές.

1.3 Μεθοδολογία

Η διαδικασία ξεκίνησε από τα τέλη του Σεπτέμβρη όπου μετά από συναντήσεις με τον επιβλέπων καθηγητή μου κ. Χρύση Γεωργίου μου ανατέθηκαν κάποια άρθρα τα οποία έπρεπε να μελετήσω και να κατανοήσω σε βάθος.

Αρχικά, μέσα από το άρθρο “Distributed Computing Column 70 Formalizing and Implementing Distributed Ledger Objects” [2] έγινε μια εισαγωγή στα Κατανεμημένα Αντικείμενα Κατάστιχων, την υπηρεσία Ατομικής Πολυεκπομπής καθώς και σε αλγορίθμους Κατανεμημένων Αντικειμένων Κατάστιχων που είναι ανθεκτικοί σε σφάλματα κατάρρευσης (Crash Fault Tolerant).

Ακολούθως μελέτησα το άρθρο “Appending Atomically in Byzantine Distributed Ledgers” [3] το οποίο εξηγεί σε βάθος την υπηρεσία Ατομικής Πολυεκπομπής που είναι ανθεκτική σε Βυζαντινά σφάλματα (BAB) αλλά και τους αλγορίθμους Κατανεμημένων Αντικειμένων Κατάστιχων που είναι ανθεκτικοί σε Βυζαντινά σφάλματα. Το πιο πάνω άρθρο συνέβαλε στην κατανόηση των ιδιοτήτων της υπηρεσίας Ατομικής Πολυεκπομπής, αλλά και τις ιδιότητες που πρέπει να πληροί ένα Κατανεμημένο Κατάστιχο για να θεωρείται ανθεκτικό σε Βυζαντινά σφάλματα.

Μετά από αυτό το άρθρο μελετήθηκε το άρθρο“Asynchronous Byzantine Distributed Ledgers” [1] που περιέχει μια πιο απλοποιημένη έκδοση του αλγορίθμου

του προηγούμενου άρθρου και ουσιαστικά είναι ο αλγόριθμος ο οποίος χρησιμοποιήθηκε στην υλοποίηση μας.

Με την μελέτη των άρθρων κατανοήθηκε το θεωρητικό υπόβαθρο σχετικά με τους Κατανεμημένους Αλγορίθμους οι οποίοι είναι ανθεκτικοί σε Βυζαντινά σφάλματα, έτσι ξεκίνησε η μελέτη της βιβλιοθήκης BFT-Smart. Αυτό έγινε σε στάδια, πρώτα διάβασα τον οδηγό χρυσής που είναι αναρτημένο στο GitHub [7], μετά διάβασα τα άρθρα που μου στάλθηκαν από τον δημιουργό της βιβλιοθήκης (“State Machine Replication for the Masses with BFT-SMART” [4] το οποίο εξηγεί σε βάθος τις δυνατότητες της βιβλιοθήκης αλλά και τον τρόπο λειτουργίας της). Ακολούθως έγινε η εγκατάσταση του BFT-Smart από το GitHub και για να σιγουρευτώ ότι λειτουργεί έτρεξα κάποια παραδείγματα όπου οι κατανεμημένοι εξυπηρετητές βρίσκονταν στην ίδια μηχανή αλλά άκουγαν σε διαφορετική πόρτα. Μετά τις δοκιμές ξεκίνησα να υλοποιώ τον αλγόριθμο τόσο για πελάτες όσο και εξυπηρετητές με την διαφορά ότι αρχικά όλα τα πειράματα έγιναν με τους εξυπηρετητές να βρίσκονται στην ίδια μηχανή αλλά να ακούνε σε διαφορετική πόρτα.

Στην συνέχεια μέσα από διάφορα διαδικτυακά σεμινάρια ξεκίνησα να μαθαίνω τις δυνατότητες του εργαλείου WindowBuilder έτσι ώστε να ξεκινήσει η δημιουργία της γραφικής διεπαφής τόσο για τους πελάτες όσο και για τους εξυπηρετητές. Μετά την ολοκλήρωση των διεπαφών για πελάτες και εξυπηρετητές ξεκίνησαν οι διαδικασίες για την διεπαφή του διαχειριστή η οποία είχε και το πιο μεγάλο επίπεδο δυσκολίας μιας και περιέχει δεδομένα από όλους τους εξυπηρετητές.

Αρχικά η διεπαφή έγινε μόνο για εξυπηρετητές που βρίσκονται στο ίδιο μηχάνημα αλλά μετά από συνεννόηση με τον επιβλέπων καθηγητή κ. Χρύση Γεωργίου αυτό επεκτάθηκε και για εξυπηρετητές σε διαφορετικά μηχανήματα. Για να επιτευχθεί αυτό έγινε μελέτη παραδειγμάτων με υποδοχές (Sockets). Τέλος έγινε έλεγχος ότι το σύστημά λειτουργεί και για εξυπηρετητές σε διαφορετικές μηχανές.

1.4 Δομή εργασίας

Κεφάλαιο 2: Στο δεύτερο κεφάλαιο περιγράφονται οι πιο σημαντικοί ορισμοί και το βασικό υπόβαθρο που πρέπει να έχει κάποιος ώστε να κατανοήσει την διπλωματική εργασία. Επίσης γίνεται μια σύντομη περιγραφή των εργαλείων που χρησιμοποιήθηκαν τόσο για την υλοποίηση των αλγορίθμων όσο και για την δημιουργία της γραφικής διεπαφής.

Κεφάλαιο 3: Στο τρίτο κεφάλαιο γίνεται μια λεπτομερής περιγραφή του αλγορίθμου που υλοποιήθηκε και των διεργασιών του πελάτη και του εξυπηρετητή. Επίσης δίνεται μεγάλη έμφαση στην Υπηρεσία Ατομικής Πολυεκπομπής που είναι ανθεκτική σε Βυζαντινά σφάλματα.

Κεφάλαιο 4: Στο τέταρτο κεφάλαιο εξηγείται σε βάθος το εργαλείο BFT- Smart, ο τρόπος λειτουργίας του, οι δυνατότητες που παρέχει και οι εγγυήσεις που διασφαλίζει.

Κεφάλαιο 5: Στο πέμπτο κεφάλαιο εξηγείται λεπτομερώς πως υλοποιήθηκαν οι κλάσεις που αποτελούν το σύστημα Κατανεμημένου Αλγορίθμου ανθεκτικού σε Βυζαντινά σφάλματα.

Κεφάλαιο 6: Στο έκτο κεφάλαιο δίνεται έμφαση στις διεπαφές του συστήματος και στις δυσκολίες που συναντήσαμε όσο αφορά τη δημιουργία τους.

Κεφάλαιο 7: Στο έβδομο κεφάλαιο παρουσιάζονται παραδείγματα εκτέλεσης που αποδεικνύουν ότι το σύστημα έχει την αναμενόμενη συμπεριφορά και ότι ο τρόπος παρουσίασης των αποτελεσμάτων είναι κατανοητός προς όλους τους χρήστες (το σύστημα είναι εύχρηστο).

Κεφάλαιο 8: Στο όγδοο κεφάλαιο κάνουμε μια σύντομη ανακεφαλαίωση, αναφέρουμε δυσκολίες που συναντήσαμε και τρόπους όπου μπορεί να βελτιωθεί το σύστημα που αναπτύχθηκε.

1.5 Γλωσσάριο

Εδώ αναφέρονται οι έννοιες που θα χρησιμοποιούμε στην ΑΔΕ μαζί με τις αγγλικές ορολογίες τους

- Ακεραιότητα – Integrity
- Αλυσίδα Κοινοποίησης – Blockchain
- Αλγόριθμος Κατανεμημένων Αντικειμένων Κατάστιχου Ανθεκτικός σε σφάλματα κατάρρευσης – Crash Fault Tolerant Algorithm
- Αντικείμενο Κατάστιχου – Ledger Object
- Αποκεντρωμένα – Decentralized
- Ασφάλεια – Safety
- Βιωσιμότητα – Livenessn
- Βυζαντινή Γραμμικοποίηση – Byzantine Linearizability (BL)
- Βυζαντινή Πληρότητα – Byzantine Completeness (BC)
- Βυζαντινό Ισχυρό Πρόθεμα – Byzantine Strong Prefix (BSP)
- Διαπροσωπεία/Διεπαφή – Interface
- Διεργασία – Process
- Εγκυρότητα – Validity
- Εξυπηρετητής – Server
- Ετικέτα – Label
- Κατανεμημένο Κατάστιχο – Distributed Ledger Object
- Κατανεμημένο Κατάστιχο Ανθεκτικό Σε Βυζαντινά Σφάλματα – Byzantine Fault Tolerant Distrubuted Ledger (BDLO)
- Κατανεμημένα Συστήματα – Distributed Systems
- Κεντρικοποιημένο – Centralized
- Λήψη – Get
- Μηχανισμός Ανοχής Σφαλμάτων – Fault Tolerance Module /Mechanism
- Πελάτης – Client
- Παραλαμβάνει – Deliver
- Πλαίσιο – Frame
- Πολυεκπομπή – Broadcast

- Πρόσθεση – Append
- Ολική Διάταξη – Total Order
- Ομοφωνία – Consensus
- Ταυτόχρονο Αντικείμενο – Concurrent Object
- Συμφωνία – Agreement
- Υπηρεσία Ατομικής Πολυεκπομπής – Atomic Broadcast Service
- Υπηρεσία Ατομικής Πολυεκπομπής Ανθεκτικής Σε Βυζαντινά Σφάλματα– Byzantine-tolerant Atomic Broadcast (BAB)

Κεφάλαιο 2

Υπόβαθρο

2.1 Κατανεμημένα Συστήματα

Με τον όρο *Κατανεμημένα Συστήματα (Distributed Systems)* εννοούμε ένα σύνολο από ανεξάρτητες μηχανές (εξυπηρετητές), οι οποίες λειτουργούν αυτόνομα δίνοντας την αίσθηση στον χρήστη ότι υπάρχουν σαν ένα ενιαίο σύστημά (δίνοντας την αίσθηση ενός κεντρικοποιημένου συστήματος). Αυτές οι μηχανές είναι ανεξάρτητες μεταξύ τους, και δεν έχουν κοινή μνήμη (πάρα μόνο την τοπική μνήμη που έχει κάθε μηχανή) έτσι ο μόνος τρόπος επικοινωνίας είναι με ανταλλαγή μηνυμάτων μέσω ενός καναλιού δικτύου επικοινωνίας. [1]

2.2 Αντικείμενα Κατάστιχου

Ένα *Αντικείμενο Κατάστιχου (Ledger Object)* είναι μια ακολουθία από εγγραφές το οποίο υποστηρίζει δύο λειτουργίες την λήψη και την πρόσθεση. Η λειτουργία λήψης επιστρέφει την ακολουθία S που είναι όλες οι εγγραφές που αποτελούν το κατάστιχο και η λειτουργία πρόσθεσης εισάγει μια νέα εγγραφή στο τέλος του κατάστιχου. Οι εγγραφές έχουν την μορφή $<r, p, v>$ όπου r είναι η ταυτότητα της εγγραφής, p είναι η

ταυτότητα της διεργασίας που πραγματοποίησε την πρόσθεση και *v* είναι τα δεδομένα της εγγραφής. [2]

Code 1 Ledger Object $\mathcal{L}$

```
1: Init: S ← ∅
2: function L.get( )
3:     return S
4: function L.append(r)
5:     S ← S||r
6:     return
```

Σχημα 2.1: Αντικείμενο Κατάστιχου [2]

2.3 Ταυτόχρονο Αντικείμενο

Ένα αντικείμενο Ο τύπου Τ ορίζεται από:

-Το σύνολο των τιμών όπου μπορεί να πάρει

-Το σύνολο των λειτουργιών όπου μια διεργασία μπορεί να χρησιμοποιήσει για να αλλάξει ή να αποκτήσει πρόσβαση στην τιμή του αντικειμένου.

Ένα αντικείμενο Ο τύπου Τ είναι *ταυτόχρονο (Concurrent)* αν είναι κοινόχρηστο και προσβάσιμο από πολλαπλές διεργασίες. Κάθε λειτουργία σε ένα αντικείμενο Ο αποτελείται από την κλήση (invoke) και την απάντηση (response), οπού η λειτουργία κλήσης προηγείται πάντα της αντίστοιχης λειτουργίας απάντησης.

Μια ιστορία από λειτουργίες στο Ο είναι μια ακολουθία από κλήσεις και απαντήσεις ξεκινώντας πάντα από μια κλήση. Η σειρά των λειτουργιών στην ιστορία καθρεφτίζει την χρονική στιγμή όπου έγιναν οι κλήσεις και οι απαντήσεις. Μια λειτουργία είναι ολοκληρωμένη σε μια ιστορία αν η ιστορία περιέχει τόσο την κλήση όσο και την απάντηση της λειτουργίας, και μια ιστορία είναι ολοκληρωμένη αν περιέχει μόνο ολοκληρωμένες λειτουργίες, διαφορετικά είναι μερική.

Μια λειτουργία π_1 *προηγείται* μιας λειτουργίας π_2 ($\pi_1 \longrightarrow \pi_2$) αν η απάντηση της π_1 εμφανίζεται πριν την κλήση της π_2, ενώ δύο λειτουργίες είναι *ταυτόχρονες* αν καμία δεν προηγείται της άλλης. [2]

2.4 Κατανεμημένα Αντικείμενα Κατάστιχου

Ένα *Κατανεμημένο Κατάστιχο (Distributed Ledger Object)* είναι ένα ταυτόχρονο αντικείμενο, που διατηρεί μια ακολουθία από εγγραφές σε ολική διάταξη (totally ordered). Η ακολουθία αρχικά είναι άδεια και υποστηρίζει δύο λειτουργίες την πρόσθεση και την λήψη. Η λειτουργία *πρόσθεση(ρ)* τοποθετεί την εγγραφή ρ στο τέλος της ακολουθίας και η λειτουργία *λήψη* επιστρέφει την ακολουθία από εγγραφές. Οι εγγραφές έχουν την μορφή $<r, p, v>$ όπου r είναι η ταυτότητα της εγγραφής, p είναι η ταυτότητα της διεργασίας που πραγματοποίησε την πρόσθεση ώστε να τοποθετηθεί η εγγραφή στο κατάστιχο και v είναι τα δεδομένα της εγγραφής.

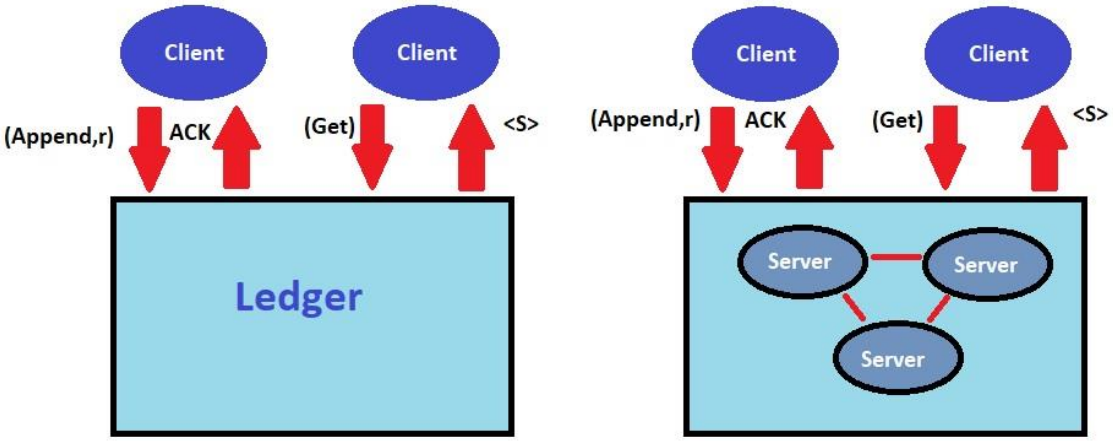
Το Κατανεμημένο Κατάστιχο υλοποιείται πάνω σε ένα σύνολο από εξυπηρετητές οι οποίοι συνεργάζονται τρέχοντας ένα κατανεμημένο αλγόριθμο (όλοι οι εξυπηρετητές τρέχουν τον ίδιο κατανεμημένο αλγόριθμο και επικοινωνούν μέσω ανταλλαγής μηνυμάτων μιας και δεν υπάρχει κοινή μνήμη).

Το Κατανεμημένο Κατάστιχο χρησιμοποιείται από ένα σύνολο από πελάτες οι οποίοι έχουν πρόσβαση σε αυτό καλώντας τις λειτουργίες πρόσθεση και λήψη, οι οποίες μεταφράζονται σε μηνύματα αιτήσεων (request) και απαντήσεων (response) που ανταλλάζονται μεταξύ των εξυπηρετητών.

Μια *εκτέλεση* είναι μια ακολουθία από κλήσεις συναρτήσεων και επιστροφές από αυτές (μια εκτέλεση ξεκινά πάντα από μια κλήση). Μια λειτουργία π είναι ολοκληρωμένη αν στην εκτέλεση ξ υπάρχει το ζεύγος από κλήση συνάρτησης και επιστροφή από την συνάρτηση (στην εκτέλεση ξ), επίσης μια λειτουργία π_1 προηγείται μιας λειτουργίας π_2 (ή μια λειτουργία π_2 ακολουθεί μια λειτουργία π_1) αν στην εκτέλεση ξ η επιστροφή από την συνάρτηση για την λειτουργία π_1 εμφανίζεται πριν από την κλήση συνάρτησης για την λειτουργία π_2 (στο ξ), διαφορετικά η π_1 και π_2 είναι ταυτόχρονες (concurrent). (Πρέπει να παρέχεται η εγγύηση ότι σε μια εκτέλεση διατηρείται η ολική σειρά ως προς τον χρόνο εκτέλεσης των λειτουργιών, δηλαδή καμία λειτουργία *λήψης* που προηγείται μιας λειτουργίας *πρόσθεση(ρ)* δεν πρέπει να επιστρέφει ακολουθία που περιέχει το ρ και όλες οι λειτουργίες *λήψης* που πραγματοποιούνται μετά την λειτουργία *πρόσθεση(ρ)* πρέπει να περιέχουν την ρ).

Με τα Αντικείμενα Κατάστιχου μπορούμε να δημιουργήσουμε ένα ασφαλές και αξιόπιστο δίκτυο από εξυπηρετητές, όπου οι πελάτες μπορούν να τοποθετούν και να

λαμβάνουν εγγραφές έχοντας την ψευδαίσθηση ότι υπάρχει μόνο μια μηχανή που διαχειρίζεται τις εγγραφές. [1] [2]



Σχήμα 2.2: Στα αριστερά υπάρχει ένα σύστημά που χρησιμοποιεί ένα Αντικείμενο Κατάστιχου (κεντρικοποιημένο) και στα δεξιά υπάρχει ένα σύστημά που χρησιμοποιεί ένα Κατανεμημένο Κατάστιχο.

2.5 Κατανεμημένο Κατάστιχο Ανθεκτικό σε Βυζαντινά Σφάλματα

Ένα *Κατανεμημένο Κατάστιχο ανθεκτικό σε Βυζαντινά σφάλματα* είναι ένα Κατανεμημένο Κατάστιχο σε ένα Βυζαντινό σύστημα. Με τον όρο Βυζαντινό σύστημα εννοούμε ένα σύστημά στο οποίο τόσο οι πελάτες όσο και οι εξυπηρετητές δύναται να υπόκεινται σε Βυζαντινά σφάλματα. Ορίζουμε τις ιδιότητες όπου ένα Κατανεμημένο Κατάστιχο πρέπει να ικανοποιεί για να προσαρμοστεί σε ένα Βυζαντινό σύστημα (να θεωρείται δηλαδή Κατανεμημένο Κατάστιχο ανθεκτικό σε Βυζαντινά σφάλματα).

Από την στιγμή που οι Βυζαντινές διεργασίες μπορούν να έχουν οποιαδήποτε συμπεριφορά (επιστρέφουν οποιαδήποτε αυθαίρετη ακολουθία ή να προσθέσουν μια οποιαδήποτε εγγραφή) θεωρούμε ότι αυτές τις ιδιότητες τις λαμβάνουν υπόψη μόνο οι ορθές διεργασίες.

- Βυζαντινή πληρότητα - Byzantine Completeness (BC): Όλες οι λειτουργίες πρόσθεσης και λήψης καλούνται από ορθούς πελάτες

- Βυζαντινό Ισχυρό Πρόθεμα - Byzantine Strong Prefix (BSP): Εάν δύο ορθοί πελάτες εκδώσουν 2 λειτουργίες λήψης όπου επιστρέφουν ακολουθίες εγγράφων S και S' διαδοχικά τότε η S είναι πρόθεμα της S' ή το αντίστροφο
- Βυζαντινή Γραμμικοποίηση - Byzantine Linearizability (BL): Έστω G το σύνολο με όλες τις ολοκληρωμένες λειτουργίες λήψεις που εκδίδονται από μη βυζαντινούς πελάτες. Έστω A το σύνολο με ολοκληρωμένες λειτουργίες πρόσθεσης της μορφής *πρόσθεση(ρ)* όπου η εγγραφή ρ επιστρέφεται στην ακολουθία S από κάποια λειτουργία λήψης που ανήκει στο σύνολο G. Η βυζαντινή γραμμικοποίηση πραγματοποιείται ως προς την σειρά εκτέλεσης πραγματικού χρόνου του συνόλου λειτουργιών G και A.

Έτσι ένα Κατανεμημένο Κατάστιχο είναι ανθεκτικό σε Βυζαντινά σφάλματα αν ικανοποιεί τις ιδιότητες BC, BSP, and BL σε ένα Βυζαντινό σύστημά. [3]

2.6 Υπηρεσία Ατομικής Πολυεκπομπής Ανθεκτική σε Βυζαντινά σφάλματα

Η υπηρεσία Ατομικής Πολυεκπομπής ανθεκτική σε Βυζαντινά σφάλματα έχει 2 βασικές λειτουργίες:

-BAB-Broadcast(m)*:* Όπου χρησιμοποιείται για αποστολή ενός μηνύματος *m* σε όλους τους εξυπηρετητές που βρίσκονται στο σύστημα.

-BAB-Deliver(m)*:* Όπου χρησιμοποιείται από την υπηρεσία Ατομικής Πολυεκπομπής για την παράδοση ενός μηνύματος m σε ένα εξυπηρετητή

Μέσω της υπηρεσίας Ατομικής Πολυεκπομπής ανθεκτική σε Βυζαντινά σφάλματα υπάρχουν εγγυήσεις για τις ακόλουθες ιδιότητες:

Εγκυρότητα (Validity): Εάν ένας μη Βυζαντινός εξυπηρετητής αποστείλει ένα μήνυμά στους άλλους εξυπηρετητές, τότε αυτό το μήνυμά σε κάποια στιγμή θα παραδοθεί.
Συμφωνία (Agreement): Εάν ένας εξυπηρετητής παραλάβει ένα μήνυμά τότε όλοι οι μη Βυζαντινοί εξυπηρετητές θα το παραλάβουν.
Ακεραιότητα (Integrity): Ένα μήνυμα μπορεί να παραληφθεί από ένα εξυπηρετητή το πολύ μια φορά εάν έχει πριν αποσταλεί από κάποιο άλλο εξυπηρετητή.

Ολική διάταξη (Total Order): Όλα τα μηνύματα πρέπει να είναι πλήρως διατεταγμένα, δηλαδή εάν ένας εξυπηρετητής παραλάβει ένα μήνυμά m πριν από κάποιο μήνυμά m' τότε όλοι οι εξυπηρετητές πρέπει να τα παραλάβουν με την ίδια σειρά.
Για την υπηρεσία ατομικής πολυεκπομπής ανθεκτική σε Βυζαντινά σφάλματα θα χρησιμοποιήσουμε την βιβλιοθήκη BFT-Smart η οποία εξηγείται σε βάθος στο επόμενο κεφάλαιο (Κεφάλαιο 4).

2.7 BFT-Smart

Το *BFT-Smart* [7] είναι μια ανοικτού πηγαίου κώδικα βιβλιοθήκη που είναι βασισμένη στην γλώσσα προγραμματισμού Java και υλοποιεί ένα ανθεκτικό σε βυζαντινά σφάλματα πρωτόκολλο με την μέθοδο αναπαραγωγής κατάστασης. Ο λόγος πού αυτή η βιβλιοθήκη διαχωρίζεται από άλλες παρόμοιες είναι επειδή παρέχει αξιοπιστία, αρθρωτότητα, αξιοποιεί πολλαπλούς πυρήνες, και η διεπαφή της είναι ευέλικτη και επεκτάσιμη.

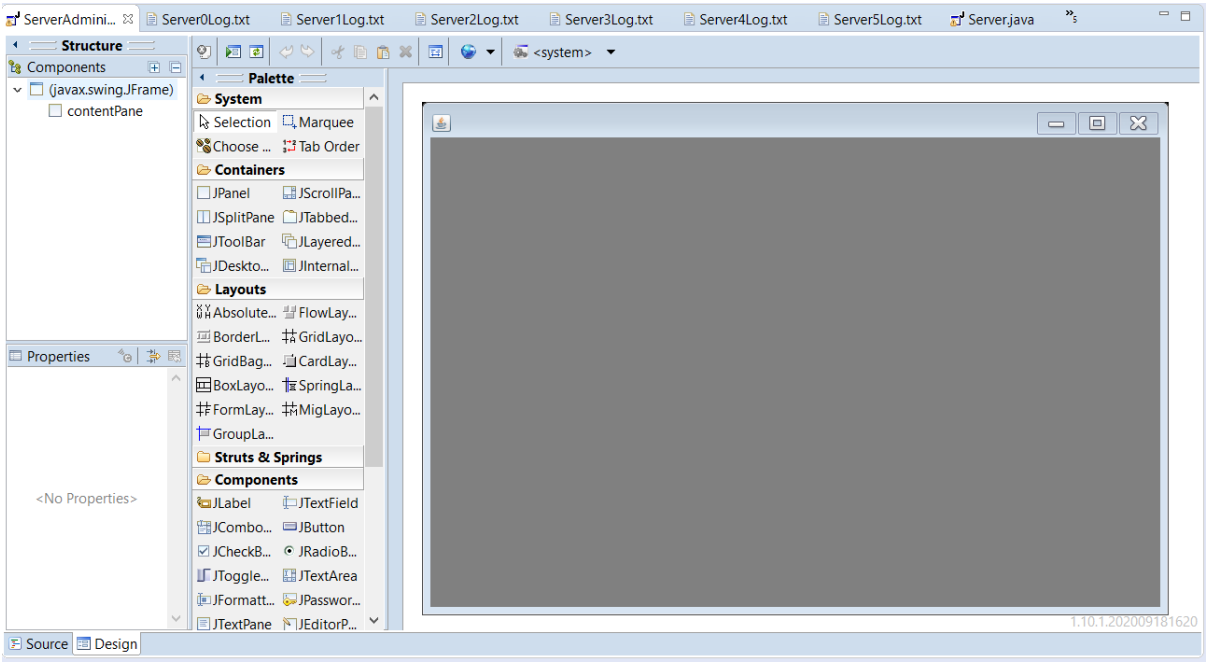
Στόχος της βιβλιοθήκης είναι η παροχή μιας διεπαφής που οι πελάτες μπορούν να επικοινωνούν με τους εξυπηρετητές μέσω της εντολής serviceProxy.invokeOrdered, οι εξυπηρετητές να λαμβάνουν το μήνυμα, να το διαχειρίζονται και να αποστέλλουν το reply στον πελάτη. Με αυτό τον τρόπο επιτυγχάνεται η δημιουργία ενός πιο απλού κώδικα ο οποίος χρησιμοποιεί τα BAB-Broadcast και BAB-Deliver μέσα από την βιβλιοθήκη BFT-Smart.

Μαζί με την εγκατάσταση της βιβλιοθήκης έρχονται και κάποια script για την αυτόματη δημιουργία του Jar αρχείου αλλά και κάποια script τόσο για το λειτουργικό σύστημα των Windows όσο και για το Linux μέσω των οποίων μπορούμε να πετύχουμε το τρέξιμο των κλάσεων. Τέλος με την εγκατάσταση παρέχονται και κάποια demo που βοηθούν στην κατανόηση της βιβλιοθήκης.

2.8 WindowBuilder

Το *WindowBuilder* [9] είναι μια βιβλιοθήκη στην γλώσσα Java η οποία δίνει την δυνατότητα στο χρήστη να δημιουργήσει διαδραστική διαπροσωπεία τόσο μέσω κώδικα όσο και μέσω γραφικού περιβάλλοντος (μέσω απλών Drag and Drop). Με την επιλογή δημιουργίας διαδραστικής διαπροσωπείας μέσω γραφικού περιβάλλοντος ο

κώδικας παράγεται αυτόματα μετά από κάθε αποθήκευση. Τα αντικείμενα τα οποία προσθέτουμε στο παράθυρο ονομάζονται “widget” και στην περίπτωση μας αρκεστήκαμε μόνο στην χρήση κουμπιών, ετικετών και παραθύρων εισόδου κειμένου. Η βιβλιοθήκη αυτή χρησιμοποιήθηκε σαν plugin στο Eclipse.



Σχήμα 2.3: WindowBuilder API κενό χρώματος γκρίζου

Κεφάλαιο 3

Αλγόριθμοι προς Υλοποίηση

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τον αλγόριθμο Κατανεμημένων Αντικειμένων Κατάστιχου ανθεκτικό σε Βυζαντινά σφάλματα (θα περιγράφουν οι διεργασίες πελάτη και εξυπηρετητή) που θα υλοποιηθεί σε γλώσσα προγραμματισμού Java με την βοήθεια της βιβλιοθήκης BFT-Smart. Η βιβλιοθήκη BFT-Smart υλοποιεί την Υπηρεσία Ατομικής Πολυεκπομπής που αναφέραμε στο προηγούμενο κεφάλαιο.

3.2 Διεργασία Πελάτη

```
1:  Init: c ← 0
2:  function L.GET( )
3:      c ← c + 1
4:      send request (c, p, GET) to at least 2f + 1 different servers
5:      wait responses (c, i, GETRESP, S) from f + 1 different servers
6:                                          carrying the same sequence S
7:      return S
8:  function L.APPEND(r)
9:      c ← c + 1
10:     send request (c, p, APPEND, r) to at least 2f + 1 different servers
11:     wait responses (c, i, APPENDRESP, ACK) from f + 1 different servers
12:     return ACK
```

Σχήμα 3.1: Κώδικας Πελάτη [1]

Αρχικά στα ματιά των πελατών το κατάστιχο δεν είναι κατανεμημένο σε ένα αριθμό από εξυπηρετητές, αλλά είναι μια ενιαία οντότητα η L. Ο αλγόριθμος που εκτελείται από τον πελάτη καλεί τις διεργασίες λήψη και πρόσθεση στο αντικείμενο

κατάστιχου. Η λειτουργία ξεκινά με την κλήση της αντίστοιχης συνάρτησης (είτε λήψης είτε πρόσθεση) και ολοκληρώνεται με την αντίστοιχη επιστροφή από την συνάρτηση. Ο αριθμός των εξυπηρετητών (n) πρέπει να είναι τουλάχιστον $3f+1$, αυτό προκύπτει από τις απαιτήσεις της υπηρεσίας ατομικής πολυεκπομπής ανθεκτική σε Βυζαντινά σφάλματα.

Ένας Βυζαντινός πελάτης μπορεί να μην ακολουθεί τον κώδικα που φαίνεται πιο πάνω αλλα να καταφέρει να προσθέσει μια λανθασμένη εγγραφή, έτσι κατά την εκτέλεση της λειτουργίας λήψης από κάποιο ορθό πελάτη αυτός μπορεί να παραλάβει μια ακολουθία από εγγραφές που περιέχει μια λανθασμένη εγγραφή.

Κατά την εκκίνηση του πελάτη γίνεται η αρχικοποίηση της μεταβλητής c, στην οποία αποθηκεύεται ο αριθμός των αιτημάτων που πραγματοποίησε ο πελάτης, αρχικά έχει τιμή 0. Ο πελάτης μπορεί να καλέσει 2 είδη διεργασιών στο αντικείμενο κατάστιχου την διεργασία λήψης και την διεργασία πρόσθεσης.

Κατά την κλήση της διεργασίας λήψης και πρόσθεσης (γραμμή 2 και 8), αρχικά αυξάνεται η μεταβλητή c κατά ένα (γραμμή 3 και 9). Μέσω της αύξησης του c επιτυγχάνεται η δημιουργία μοναδικής ταυτότητας για το αίτημα. Ακολούθως ο πελάτης στέλνει το αίτημα του, μαζί με το αναγνωριστικό του και το αναγνωριστικό αιτήματος (σε περίπτωση αιτήματος πρόσθεσης και την εγγραφή που θέλει να προσθέσει) σε τουλάχιστον $2f+1$ εξυπηρετητές (γραμμή 4 και 10) όπου f ο αριθμός των Βυζαντινών εξυπηρετητών (εμείς επιλέξαμε να το στέλνουμε σε όλους και αφού $n>=3f+1$ η συνθήκη ικανοποιείται). Ο λόγος που επιλέξαμε $2f+1$ είναι για να σιγουρευτούμε ότι τουλάχιστον ένας εξυπηρετητής θα στείλει το μήνυμα στους υπολοίπους και ότι θα λάβουμε $f+1$ ίδιες απαντήσεις. (Αν λάβουμε $f+1$ ίδιες απαντήσεις τότε μια εξ ’αυτών είναι από ορθό εξυπηρετητή άρα η απάντηση που έλαβε ο πελάτης είναι ορθή).

Μια λειτουργία κλήσης είναι ολοκληρωμένη αν ο πελάτης λάβει τουλαχιστον $f+1$ ίδιες απαντήσεις όπου περιέχουν την ίδια ακολουθία από εγγραφές και μια λειτουργία πρόσθεσης είναι ολοκληρωμένη αν ο πελάτης λάβει τουλάχιστον $f+1$ ACK από διαφορετικούς εξυπηρετητές (γραμμή 5 και 11). Και οι δύο περιπτώσεις εγγυούνται ότι ο πελάτης πήρε απάντηση από τουλάχιστον ένα σωστό εξυπηρετητή.

(Επειδή η υπηρεσία ατομικής πολυεκπομπής ανθεκτικής σε Βυζαντινά σφάλματα απαιτεί τουλάχιστον $3f+1$ εξυπηρετητές, στην υλοποίηση λαμβάνουμε τουλάχιστον $2f+1$ ίδιες απαντήσεις που είναι μεγαλύτερο από το $f+1$ ($2f+1>f+1$) άρα η συνθήκη

ικανοποιείται, ο λόγος που λαμβάνουμε 2*f*+1 είναι επειδή έχουμε τουλάχιστον 3*f*+1 εξυπηρετητές όπου οι *f* απαντούν Βυζαντινά αρά 3*f*+1-*f*=2*f*+1)

3.3 Διεργασία Εξυπηρετητή (Αλγόριθμος)

```
1:  Init: S_i ← ∅
2:  receive (c, p, GET) from process p
3:      BAB-broadcast(c, p, GET, i)
4:  upon (BAB-deliver(c, p, GET, j)) do
5:      send response (c, i, GETRESP, S_i) to p
6:  receive (c, p, APPEND, r) from process p
7:      BAB-broadcast(c, p, APPEND, r, i)
8:  upon (BAB-deliver(c, p, APPEND, r, j)) do
9:      if (r ∉ S_i) then
10:         S_i ← S_i || r
11:         send resp. (c, i, APPENDRESP, ACK) to p
```

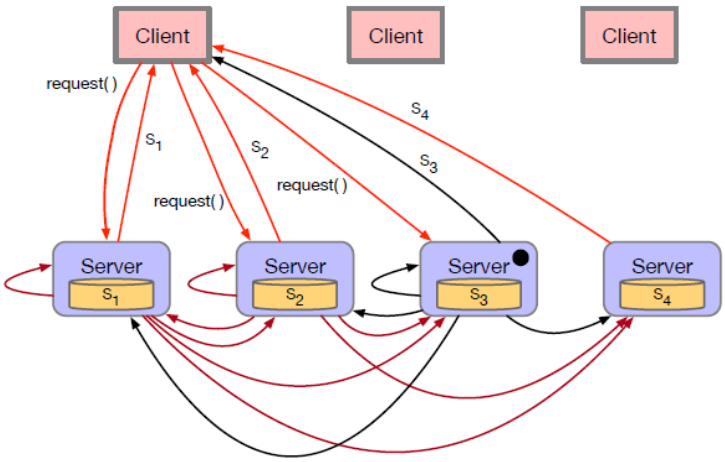
Σχήμα 3.2: Κώδικας Εξυπηρετητή [1]

Ο αλγόριθμος που εκτελείται από τους εξυπηρετητές χρησιμοποιεί την υπηρεσία Ατομικής Πολυεκπομπής ανθεκτική σε Βυζαντινά σφάλματα. Μέσω της υπηρεσίας επιβάλλεται ολική διάταξη στα μηνύματα που στέλνονται στους εξυπηρετητές.

Αρχικά πριν ξεκινήσει η εξυπηρέτηση αιτημάτων αρχικοποιείται το προσωπικό αντίγραφο του αντικείμενου κατάστιχου του εξυπηρετητή σε μια άδεια λίστα (Γραμμή 1). Ακολούθως μόλις ληφθεί κάποιο αίτημα (Γραμμή 2 και 6) αυτό στέλνεται και στους υπόλοιπους εξυπηρετητές (BAB-Broadcast στην γραμμή 3 και 7). Στην συνέχεια αυτοί το παραλαμβάνουν (BAB-Deliver στην γραμμή 4 και 8) και το επεξεργάζονται, στην περίπτωση που το αίτημα είναι αίτημα λήψης στέλνουν το αντικείμενο κατάστιχου τους (Γραμμή 5). Στην περίπτωση που είναι αίτημα πρόσθεσης, προσθέτουν την εγγραφή στην τελευταία θέση του αντικείμενου κατάστιχου τους (γραμμή 10) και στέλνουν ACK στον πελάτη (γραμμή 11). Η υπηρεσία Ατομικής Πολυεκπομπής ανθεκτικής σε Βυζαντινά σφάλματα εγγυάται ότι όλοι οι σωστοί εξυπηρετητές θα λάβουν την ίδια ακολουθία από αιτήματα με την ίδια σειρά. Θα επεξεργαστούν τις λειτουργίες με την ίδια σειρά διατηρώντας έτσι την κατάσταση τους σταθερή. Στο [1] αποδεικνύεται η ορθότητα του αλγορίθμου.

Στο Σχήμα 3.3 φαίνεται και γραφικά η διαδικασία επικοινωνίας μεταξύ πελατών και εξυπηρετητών. Πιο συγκεκριμένα όταν ένας πελάτης θέλει να πραγματοποιήσει κάποιο αίτημα τότε πρέπει να στείλει αυτό το αίτημα σε τουλάχιστον $2f+1$ εξυπηρετητές. Στην Σχήμα 3.3 έχουμε ένα Βυζαντινό εξυπηρετητή και 3 μη Βυζαντινούς. Αυτό ικανοποιεί όλες της απαιτήσεις για να λειτουργήσει το σύστημα αφού έχουμε $n>=3f+1$.

Όταν ο πελάτης στείλει το αίτημα σε $2f+1$ εξυπηρετητές, αυτοί το στέλνουν μεταξύ τους (BAB-Broadcast) και το παραλαμβάνουν (BAB-Deliver). Ακολούθως το επεξεργάζονται με τον τρόπο που εξηγήθηκε πιο πάνω και στέλνουν την απάντηση τους στον πελάτη. Ο πελάτης έλαβε 3 ίδιες απαντήσεις από ορθούς εξυπηρετητές και μια λανθασμένη από τον Βυζαντινό. Επειδή έλαβε τουλάχιστον $f+1$ ίδιες απαντήσεις σημαίνει πως κάποιος ορθός εξυπηρετητής απάντησε και η απάντηση που έλαβε ο πελάτης είναι ορθή.



Σχήμα 3.3: Οπτική αναπαράσταση λειτουργίας αλγορίθμου

Κεφάλαιο 4

BFT-Smart

4.1 Εισαγωγή

Το BFT-Smart [7] είναι μια ανοικτού πηγαίου κώδικα βιβλιοθήκη γραμμένη στη γλώσσα προγραμματισμού JAVA που υλοποιεί ένα ανθεκτικό σε Βυζαντινά σφάλματα πρωτόκολλο με την μέθοδο αναπαραγωγής κατάστασης (state machine replication). Κάποια από τα σημαντικότερα χαρακτηριστικά πού διαφοροποιούν την βιβλιοθήκη από άλλες παρόμοιες είναι η βελτιωμένη αξιοπιστία, αρθρωτότητα, η δυνατότητα αξιοποίησης πολυπύρηνων συστημάτων και η ευέλικτη προγραμματιστική διεπαφή.

Όταν κατεβάζουμε την βιβλιοθήκη, κατεβάζουμε το πηγαίο κώδικα (src/), τα jar file, τα διάφορα dependencies (lib/), τα configuration file (config/), τα scripts οπού χρησιμοποιούνται για το τρέξιμο της βιβλιοθήκης (runscripts/) και το documentation (doc/). Τέλος απαιτείται JRE έκδοσης 1.8 ή πιο πρόσφατη.

Η μεταγλώττιση γίνεται μέσω του εργαλείου ant το οποίο παράγει το jar file, αυτό επιτυγχάνεται μέσω της εντολής ant, ενώ για να τρέξουμε κάποια κλάση χρησιμοποιούμε το:

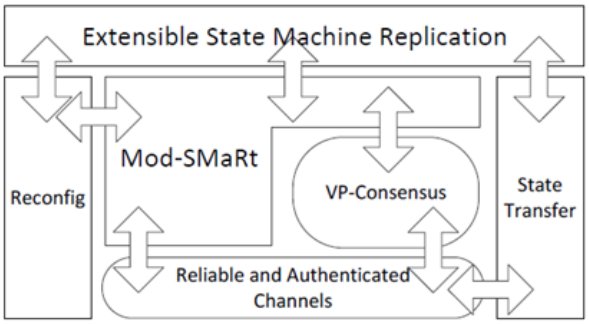
- ./runscripts/smartrun.sh bftsmart/demo/blockchain/BFServer 0 192.168.10.1
  Για να εκτελέσουμε το Βυζαντινό εξυπηρετητή με id το 0 και IP διεύθυνση διαχειριστή το 192.168.10.1
- /runscripts/smartrun.sh bftsmart/demo/blockchain/BServer 2 192.168.10.1
  Για να εκτελέσουμε ένα μη Βυζαντινό εξυπηρετητή με id το 2 και IP διεύθυνση διαχειριστή το 192.168.10.1
- ./runscripts/smartrun.sh bftsmart/demo/blockchain/ClientGui 1
  Για να εκτελέσουμε τον πελάτη με id το 1
- /runscripts/smartrun.sh bftsmart/demo/blockchain/ServerAdministrator5Server 7
  Για να εκτελέσουμε τον διαχειριστή για 7 εξυπηρετητές.

Περισσότερες πληροφορίες στο Παράρτημα Ε.

4.2 Βασικές Αρχές Σχεδίασης

Από προεπιλογή το BFT-Smart ανέχεται μη κακοήθη Βυζαντινά σφάλματα σε ρεαλιστικό σύστημα όπου (i) τα μηνύματα μπορούν να καθυστερήσουν ή ακόμα και να καταστραφούν, (ii) οι διεργασίες μπορούν να εκτελούν μη συνήθη και Βυζαντινή συμπεριφορά. Για να είναι πιο ανθεκτικό σε Βυζαντινές συμπεριφορές, το BFT-Smart παρέχει κρυπτογραφικές υπογραφές (για βελτιωμένη ανοχή σε Βυζαντινά σφάλματα).

Το BFT-Smart δίνει τρομερή έμφαση στην ορθότητα και την απλότητα, αυτό έχει σαν αποτέλεσμα να αποφεύγεται επιπλέον πολυπλοκότητα η οποία μπορεί να οδηγήσει σε βελτιώσεις από πλευράς απόδοσης αλλά παράλληλα να δημιουργήσει προβλήματα στην εύκολη ανάπτυξη του και την διατήρηση της ορθότητας του. Για αυτό τον λόγο το πρωτόκολλο παραμένει απλό αποφεύγοντας τεχνικές όπως εικασία (speculation) και διοχέτευση (pipeline). Επίσης το BFT-Smart δίνει έμφαση στην αρθρωτότητα αφού υλοποιεί ένα αρθρωτό πρωτόκολλο που η ομοφωνία (consensus) βρίσκεται στον πυρήνα (διαφοροποιώντας την από άλλες μονάδες), αυτή η αρθρωτότητα επιτυγχάνει ευκολία στην υλοποίηση του πρωτοκόλλου. Πέρα από την ομοφωνία άλλες σημαντικές μονάδες είναι η μονάδα μεταφοράς κατάστασης (state transfer) και η επαναδιαμόρφωση (είναι εντελώς διαφοροποιημένες από τις άλλες μονάδες).



Σχήμα 4.1: Αρθρωτότητα BFT-SMART [4]

Το BFT-Smart θέλει να επιτύχει ευκολία στην ανάπτυξη εφαρμογών, έτσι ενθυλακώνει όλη την πολυπλοκότητα του σε ένα απλό API που μπορεί να χρησιμοποιηθεί από τους προγραμματιστές. Πιο συγκεκριμένα, ο πελάτης μπορεί να καλέσει μια μέθοδο (εντολή) η οποία στέλνει οδηγίες στους εξυπηρετητές. Οι εξυπηρετητές (έχουν υλοποιήσει αυτές τις εντολές σε μεθόδους) εκτελούν αυτές τις εντολές. (Η διαδικασία που περιεγράφηκε πιο πάνω είναι γνωστή και ως SMR προγραμματιστικό μοντέλο). Τέλος, το BFT-Smart θέλει να εκμεταλλευτεί την πολυπύρηνη αρχιτεκτονική των εξυπηρετητών για να βελτιώσει κάποιες ακριβές δουλείες επεξεργασίας στο κρίσιμο μονοπάτι του πρωτοκόλλου έτσι γίνεται χρήση ενός αριθμού νημάτων που υποστηρίζεται από τους εξυπηρετητές.

4.3 Μοντέλο Συστήματος

Το BFT-SMART για να είναι ανθεκτικό σε Βυζαντινά σφάλματα χρειάζεται $n>=3f+1$ εξυπηρετητές όπου f ο αριθμός των Βυζαντινών εξυπηρετητών, και μπορεί να δουλέψει για μη περιορισμένο αριθμό από Βυζαντινούς πελάτες, αν ικανοποιούνται οι πιο πάνω συνθήκες τότε παρέχει βιωσιμότητα (η εγγύηση ότι τίποτα κακό δεν θα συμβεί).

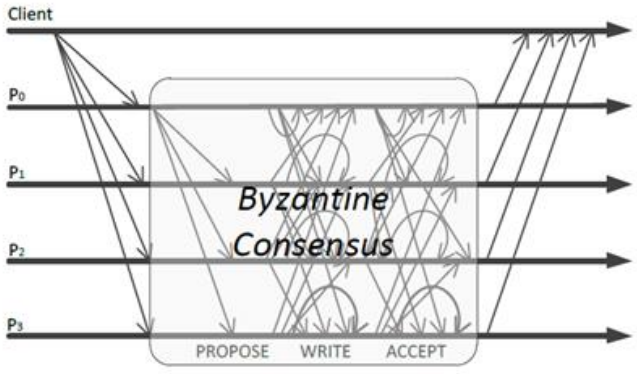
Επίσης παρέχει αξιόπιστες συνδέσεις σημείου-προς-σημείου για επικοινωνία μεταξύ διεργασιών. Αυτές οι συνδέσεις υλοποιούνται χρησιμοποιώντας κωδικούς αυθεντικοποιήσης μηνυμάτων (MAC) πάνω σε TCP/IP. Τα συμμετρικά κλειδιά για τα κανάλια επικοινωνίας μεταξύ εξυπηρετητών δημιουργούνται μέσα από τον αλγόριθμο Diffie-Hellman χρησιμοποιώντας ένα ζεύγος RSA κλειδιών για κάθε εξυπηρετητή. Τα κλειδιά για τα κανάλια επικοινωνίας πελάτη εξυπηρετητή δημιουργούνται με βάση τις

ταυτότητες τους, χωρίς να χρειάζεται ο πελάτης να κρατά το ζεύγος κλειδιών. Παρόλα αυτά αν θέλουμε μπορούμε να ενεργοποιήσουμε και την αυθεντικοποιήση για πελάτες αν ενεργοποιήσουμε την επιλογή υπογεγραμμένων αιτημάτων (στο system.comfiq).

4.4 Βασικά Πρωτόκολλα

Σε αυτό το υποκεφάλαιο θα κάνουμε αναφορά στα πρωτόκολλα που παρέχονται από το BFT-SMART τα οποία χρησιμοποιήθηκαν στην διπλωματική.

Ολική διάταξη πολλαπλής εκπομπής (Total Order Multicast): Αυτό το πρωτόκολλο επιτυγχάνεται χρησιμοποιώντας το Mod-Smart, ένα αρθρωτό πρωτόκολλο που υλοποιεί το BFT-Smart χρησιμοποιώντας μια υποκειμενική ομοφωνία (underlying consensus).Πιο συγκεκριμένα γίνεται χρήση μιας επέκτασης του αλγορίθμου Βυζαντινής ομοφωνίας οδηγούμενης από ένα ηγέτη. Κατά την διάρκεια της κανονικής εκτέλεσης οι πελάτες στέλνουν αιτήματα σε όλους τους εξυπηρετητές και αναμένουν για την απάντηση τους. Η ολική διάταξη επιτυγχάνεται μέσα από μια ακολουθία από πρότυπα ομοφωνίας, όπου το κάθε πρότυπο αποφασίζει για μια δέσμη από αιτήματα πελάτη, και κάθε πρότυπο αποτελείται από 3 στάδιά επικοινωνίας. Στο πρώτο στάδιο ο ηγέτης (ένας εξυπηρετητής) στέλνει μήνυμα ΠΡΟΤΑΣΗΣ στους υπολοίπους εξυπηρετητές, όπου αυτό το μήνυμα περιέχει όλα τα αιτήματα που θα επεξεργαστούν. Μετά ακολουθούν τα στάδιά ΓΡΑΨΙΜΟ και ΑΠΟΔΟΧΗ, δύο στάδιά όπου όλοι οι εξυπηρετητές επικοινωνούν μεταξύ τους. Τόσο το ΓΡΑΨΙΜΟ όσο και η ΑΠΟΔΟΧΗ περιέχουν το cryptographic hash του συνόλου των αιτημάτων.



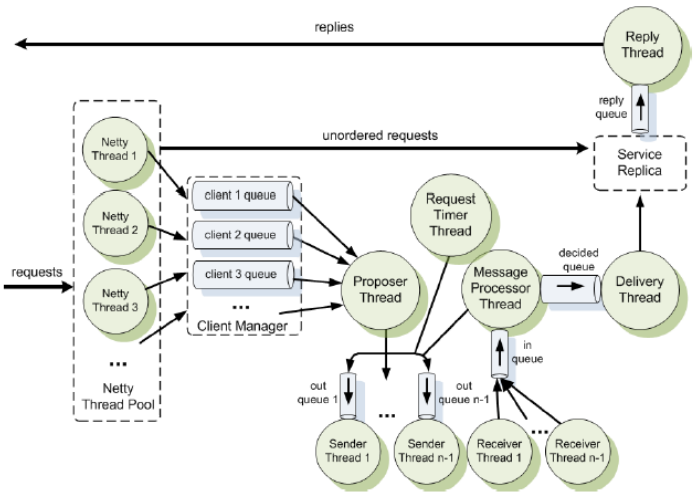
Σχήμα 4.2: Πρωτόκολλο Ολικής Διάταξης Πολυεκπομπής [4]

Μεταβίβαση Κατάστασης: Για να υλοποιήσουμε ένα σύστημα με χρήση της μεθόδου Αναπαραγωγής Κατάστασης, οι εξυπηρετητές θα πρέπει να είναι σε θέση να επισκευαστούν και να επανενταχθούν στο σύστημα χωρίς να χρειαστεί να ξεκινήσει από την αρχή ολόκληρη η υπηρεσία. Πιο αναλυτικά σε περίπτωση που κάποιος εξυπηρετητής απενεργοποιηθεί θα πρέπει να είναι σε θέση όταν επισκευαστεί να μπορεί να επανενταχθεί στο σύστημα και να έχει την πιο πρόσφατη κατάσταση του συστήματος, για να πραγματοποιηθεί αυτό απαιτείται μια σταθερή μονάδα αποθήκευσης.

Οι τεχνικές που χρησιμοποιούνται για την ανάρρωση των εξυπηρετητών είναι οι ακόλουθες:

1) Αποθηκεύονται δέσμες από λειτουργίες στο δίσκο καθώς αυτές οι λειτουργίες εκτελούνται από την υπηρεσία.
2) Λαμβάνουμε snapshot από διαφορετικά σημεία εκτέλεσης, από διαφορετικούς εξυπηρετητές για να μην σταματήσουμε το σύστημα (αν λάμβαναν snapshot όλοι οι εξυπηρετητές ταυτόχρονα τότε θα σταματούσε προσωρινά η λειτουργία του συστήματος)
3) Πραγματοποιούμε μεταφορά κατάστασης με συνεργατικό τρόπο, όπου κάθε εξυπηρετητής στέλνει διαφορετικό μέρος της κατάστασης που υπάρχει στο σύστημα στον εξυπηρετητή που αναρρώνει.

4.5 Υλοποίηση



Σχήμα 4.3: Υλοποίηση BFT-Smart [4]

Το Σχήμα 4.3 παρουσιάζει την αρχιτεκτονική που ακολουθείται για την επεξεργασία μηνυμάτων και τον τρόπο που υλοποιήθηκε το πρωτόκολλο.

Τα αιτήματα του πελάτη λαμβάνονται μέσα από ένα σύνολο νημάτων που παρέχονται από την επικοινωνία Netty. Αυτή η σχεδίαση επιτρέπει στο BFT-Smart να διαχειρίζεται χιλιάδες συνδέσεις πελάτη εξυπηρετητή. Μόλις φθάσει το μήνυμα ενός πελάτη ελέγχεται αν ο πελάτης αιτείται να εκτελεσθεί το μήνυμα (αίτημα) ως προς την πραγματική ώρα ή όχι (εμάς τα αιτήματα μας είναι μόνο ως προς την πραγματική ώρα άρα θα εξηγήσουμε μόνο αυτή την περίπτωση).Σε περίπτωση που το μήνυμα είναι με βάση την πραγματική ώρα τότε φθάνει στον διαχειριστή πελάτων (ένα σύνολο από ουρές μια για κάθε πελάτη), στο διαχειριστή πελατών ελέγχεται η ακεραιότητα του αιτήματος και προστίθεται στην αντίστοιχη ουρά του πελάτη (πχ, αν το αίτημά το έκδωσε ο πελάτης 1 θα αποθηκευτεί στην ουρά 1).

Ακολούθως το νήμα ΠΡΟΤΑΣΗΣ δημιουργεί την δέσμη με τα αιτήματα και στέλνει το μήνυμα ΠΡΟΤΑΣΗΣ για την τρέχων ομοφωνία πρωτοκόλλου (η δέσμη γεμίζει με αιτήματα μέχρι να φθάσουμε το μέγιστο αριθμό αιτημάτων ή μέχρι να μην υπάρχουν άλλα αιτήματα). Το νήμα ΠΡΟΤΑΣΗΣ είναι ενεργοποιημένο μόνο στον ηγέτη εξυπηρετητή ο οποίος είναι και ο μόνος που εκδίδει το μήνυμα ΠΡΟΤΑΣΗΣ. Το μήνυμα ΠΡΟΤΑΣΗΣ έχει προορισμό τους άλλους εξυπηρετητές, και ο κάθε εξυπηρετητής το λαμβάνει μέσα από την ουρά OUT.

Κάθε μήνυμα που στέλνεται από ένα εξυπηρετητή σε κάποιο άλλο τοποθετείται στην ουρά OUT, από την οποία το νήμα αποστολέας λαμβάνει το μήνυμα το σειριοποιεί, παράγει το MAC (και το τοποθετεί στο μήνυμα) και το στέλνει μέσω TCP υποδοχής (Socket). Για κάθε νήμα αποστολέα ο κάθε εξυπηρετητής έχει και ένα νήμα παραλήπτη το οποίο διαβάζει το μήνυμα, πραγματοποιεί τον έλεγχο αυθεντικοποίησης, το αποσειριοποιεί, και το βάζει στην IN ουρά. Στην ουρά IN τοποθετούνται τα μηνύματα που λήφθηκαν από άλλους εξυπηρετητές με σκοπό να επεξεργαστούν.

Το νήμα επεξεργασίας μηνυμάτων είναι υπεύθυνο για την επεξεργασία μηνυμάτων. Αυτό το νήμα παίρνει τα μηνύματα από την IN ουρά, αν ανήκουν στην ομοφωνία που εκτελείται την δεδομένη στιγμή τα διαχειρίζεται, διαφορετικά αν ανήκουν σε μια μελλοντική ομοφωνία θα επεξεργαστούν στο μέλλον όταν έρθει η σειρά αυτής της ομοφωνίας, διαφορετικά απορρίπτονται.

Όταν η τρέχουσα ομοφωνία ολοκληρωθεί η δέσμη με τα αιτήματα τοποθετείται στην ουρά απόφασης. Το νήμα διανομής λαμβάνει δέσμες από την ουρά απόφασης,

αποσειριοποιεί όλα τα αιτήματα, τα απομακρύνει από την αντίστοιχη ουρά του πελάτη και μαρκάρει την τρέχων ομοφωνία ως ολοκληρωμένη. Μετά το νήμα διανομής καλεί τον εξυπηρετητή υπηρεσίας (Service Replica) ο οποίος εκτελεί το αίτημα και δημιουργεί τις αντίστοιχες απαντήσεις (replies). Όταν δημιουργήσει την απάντηση για κάποιο αίτημα το προσθέτει στην ουρά απάντησης. Το νήμα απάντησης παίρνει τις απαντήσεις από την ουρά απάντησης και τις στέλνει στον αντίστοιχο πελάτη.

4.6 Εναλλακτικές Διαμορφώσεις

Όπως αναφέραμε και στην αρχή του κεφαλαίου από προεπιλογή το BFT-Smart ανέχεται μη κακόβουλα Βυζαντινά σφάλματα, παρόλα αυτά μέσω αλλαγής των παραμέτρων στο config.system μπορεί να υποστηρίζει και δύο άλλα είδη λαθών.

Ανθεκτικό σε σφάλματα κατάρευσης: Το BFT-Smart υποστηρίζει μια παράμετρο διαμόρφωσης (system.bft) που αν ενεργοποιηθεί κάνει το σύστημα αυστηρό σε σφάλματα κατάρρευσης. Όταν η παράμετρος ενεργοποιηθεί το σύστημα ανέχεται f εξυπηρετητές όπου μπορεί να σταματήσουν την εκτέλεση τους και συνολικό αριθμό εξυπηρετητών $n>=2f+1$. Επίσης όταν ενεργοποιηθεί αυτή η παράμετρος αγνοείται το στάδιο ΓΡΑΨΙΜΟ. Τα υπόλοιπα είναι τα ίδια με την ανθεκτική σε Βυζαντινά σφάλματα εκτέλεση.

Ανθεκτικό σε κακόβουλα Βυζαντινά σφάλματα: Κατά την ενεργοποίηση αυτής της παραμέτρου στο confiq.system γίνεται χρήση υπογραφής δημόσιου κλειδιού στα αιτήματα όπου πραγματοποιούν οι πελάτες, αυτό έχει σαν αποτέλεσμα οι πελάτες να μην μπορούν να πλαστογραφήσουν τον κώδικα αυθεντικοποίησης μηνύματος (MAC) που χρησιμοποιείται κατά την επικοινωνία μεταξύ εξυπηρετητών.

4.7 Διεπαφές Προγράμματος

Στην διπλωματική εργασία υλοποιούμε τις κλάσεις του πελάτη και του εξυπηρετητή έτσι χρησιμοποιήσαμε την διεπαφή του πελάτη και του εξυπηρετητή για να τις υλοποιήσουμε.

Διεπαφή Πελάτη: Για την χρήση της διεπαφής του πελάτη έγινε εισαγωγή (import) του navigators.smart.tom.ServiceProxy το οποίο περιέχει τις μεθόδους που ο πελάτης χρησιμοποιεί για να στείλει μηνύματα στους εξυπηρετητές και να λάβει απαντήσεις από αυτούς. Γίνετε χρήση της συνάρτησης invokeOrdered(byte[] request) η οποία στέλνει μήνυμα σε όλους τους εξυπηρετητές χρησιμοποιώντας το πρωτόκολλο ολικής διάταξης πολλαπλής εκπομπής. Ακολούθως αφού συγκρίνει τις απαντήσεις από κάθε εξυπηρετητή, στέλνει την απάντηση της απαρτίας των εξυπηρετητών στον πελάτη. Ο πελάτης μέχρι να λάβει την απάντηση δεν μπορεί να στείλει άλλα αιτήματα.

Διεπαφή Εξυπηρετητή: Για την χρήση της διεπαφής του εξυπηρετητή έγινε εισαγωγή του navigators.smart.tom.ServiceReplica το οποίο περιέχει τις συναρτήσεις για να λαμβάνουν οι εξυπηρετητές τα αιτήματα των πελατών και να τα επεξεργάζονται. Η κλάση ServiceReplica παρέχει 4 κατασκευαστές (constructors) που αρχικοποιούνται χρησιμοποιώντας την υλοποίηση της εφαρμογής του εξυπηρετητή. Αυτοί οι κατασκευαστές έχουν σαν παράμετρο τις διεπαφές που υλοποιούνται από τον κώδικα της εφαρμογής. Οι διεπαφές είναι:

- Executable: Το Executable επιτρέπει στην εφαρμογή να αποφασίσει πως θα εκτελέσει το μήνυμα που έλαβε.
- Recoverable: Ορίζει πως θα πραγματοποιηθεί η μεταφορά της κατάστασης, η εφαρμογή συνήθως σειριοποιεί (serialize) την κατάσταση.
- Replier: Η εφαρμογή αποφασίζει πως θα απαντά στα αιτήματα που έλαβε από τους πελάτες.

Πέρα από την κλάση ServiceReplica η οποία είναι υπεύθυνη για την επικοινωνία μεταξύ εξυπηρετητών και πελατών, στην διεπαφή εξυπηρετητή πρέπει να υλοποιηθεί και η συνάρτηση appExecuteOrdered μέσα στην οποία ο κάθε εξυπηρετητής λαμβάνει τα αιτήματα των πελατών, τα επεξεργάζεται και στέλνει την απάντηση του σε αυτούς.

4.8 Διαμόρφωση BFT-Smart

Οι παράμετροι για να διαμορφώσουμε τους εξυπηρετητές του BFT-Smart είναι αποθηκευμένοι σε 2 αρχεία, το hosts.config και στο system.config.

Στο host.config βρίσκονται οι διευθύνσεις και οι πόρτες κάθε εξυπηρετητή. Ο αριθμός των εξυπηρετητών πρέπει να είναι ίδιος με τον αριθμό που ορίζεται στην παράμετρο system.servers.num στο system.config. Στην πρώτη στήλη ορίζεται η ταυτότητα του εξυπηρετητή, στην δεύτερη στήλη ορίζεται η IP διεύθυνση του, στην τρίτη στήλη είναι η πόρτα που επικοινωνεί με τους πελάτες και στην τέταρτη η πόρτα που επικοινωνεί με τους εξυπηρετητές.

```
0 127.0.0.1 11000 11001
1 127.0.0.1 11010 11011
2 127.0.0.1 11020 11021
3 127.0.0.1 11030 11031
4 127.0.0.1 11040 11041
```

Σχήμα 4.4: Το hosts.config αρχείο

Στο system.config υπάρχουν πολλές επιλογές από τις οποίες μπορούμε να διαμορφώσουμε το πρωτόκολλο, αφού υπάρχουν επιλογές όπως, τον αριθμό των εξυπηρετητών, τον αριθμό των Βυζαντινών εξυπηρετητών κτλ, οι παράμετροι περιγράφονται ακολούθως.

Όνομα Παραμέτρου	Περιγραφή
`system.servers.num`	Ορίζει τον αριθμό των εξυπηρετητών στο σύστημα
`system.servers.f`	Ορίζει τον αριθμό των Βυζαντινών εξυπηρετητών
`system.communication.useSenderThread`	Ορίζει αν θα γίνεται χρήση νημάτων για την αποστολή δεδομένων
`system.communication.inQueueSize`	Ορίζει τον αριθμό μηνυμάτων που μπορούν να αποθηκευτούν στην ουρά IN
`system.communication.outQueueSize`	Ορίζει τον αριθμό μηνυμάτων που μπορούν να αποθηκευτούν

	στην ουρά OUT
system.communication.useSignatures	Ορίζει αν οι πελάτες πρέπει να χρησιμοποιούν υπογραφή στα MAC
system.communication.useMACs	Ορίζει αν οι εξυπηρετητές πρέπει να πραγματοποιούν αυθεντικοποίηση στην μεταξύ τους επικοινωνία
system.communication.defaultkeys	Ορίζει αν οι διεργασίες θα χρησιμοποιούν το ίδιο ζεύγος δημοσίου ιδιωτικού κλειδιού μεταξύ τους
system.communication.bindaddress	Ορίζει το IP address όπου ο εξυπηρετητής θα κάνει bind. Αν σε αυτή την παράμετρο δεν ορισθεί μια σωστή διεύθυνση τότε ο εξυπηρετητής θα κάνει bind στην διεύθυνση της μηχανής. Αν το config/host.config ορίζει loopback address τότε αυτή η παράμετρος γίνεται override από αυτή την διεύθυνση.
system.communication.hashAlgorithm	Ορίζει αλγόριθμο που χρησιμοποιείται για υπολογισμό των hashes
system.communication.hmacAlgorithm	Ορίζει αλγόριθμο που χρησιμοποιείται για αυθεντικοποίηση των μηνυμάτων μεταξύ διεργασιών.
system.communication.secretKeyAlgorithm	Ορίζει τον αλγόριθμο που χρησιμοποιείται για τη

	δημιουργία των μυστικών κλειδιών που δημιουργούν τα MAC.
system.communication.signatureAlgorithm	Ορίζει αλγόριθμο που επαληθεύει τα αιτήματα των πελατών.
system.communication.hmacAlgorithmProvider	Ορίζει τον αλγόριθμο HMAC.
system.communication.secretKeyAlgorithmProvider	Ορίζει τον αλγόριθμο για την δημιουργία μυστικών κλειδιών.
system.communication.signatureAlgorithmProvide	Ορίζει τον αλγόριθμο μυστικού κλειδιού.
system.communication.hashAlgorithmProvider	Ορίζει τον hash αλγόριθμο.
system.totalordermulticast.timeout	Ο χρόνος που αναμένει κάποιος εξυπηρετητής μέχρι να λάβει το μήνυμα ΠΡΟΤΑΣΗΣ. Αν δεν το λάβει, τότε καλείται το πρωτόκολλο αλλαγής ηγέτη επειδή θεωρείται ότι ο ηγέτης είναι Βυζαντινός.
system.totalordermulticast.highMark	Μέγιστος αριθμός μηνυμάτων που δεν απορρίπτονται. Όταν ένας εξυπηρετητής καθυστερεί, αποθηκεύει την ομοφωνία σε μια ουρά “out of context”. Αν ο αριθμός της ομοφωνίας που επεξεργάζεται εκείνη την στιγμή είναι πιο μεγάλη από την τιμή αυτής της παραμέτρου, τότε ο εξυπηρετητής απορρίπτει τα μηνύματα και καλεί το πρωτόκολλο μεταφοράς κατάστασης.

system.totalordermulticast.maxtachsize	Ορίζει τον μέγιστο αριθμό μηνυμάτων που μπορεί να συμπεριληφθεί σε ένα μήνυμα πρότασης
system.totalordermulticast.nonces	Ορίζει τον αριθμό μη ντετερμινιστικών ενεργειών.
system.totalordermulticast.state_transfer	Ενεργοποιεί το πρωτόκολλο μεταφοράς κατάστασης.
system.totalordermulticast.checkpoint_period	Ορίζει κάθε πόσο ο εξυπηρετητής ζητά την κατάσταση της εφαρμογής έτσι ώστε να καθαρίσει το log file του.
system.totalordermulticast.revival_highMark	Μέγιστο μήνυμα (maximum ahead of time) που δεν απορρίπτεται από την ώρα που εξυπηρετητής μπήκε στο σύστημα.
system.initial.view	Είναι τα id όλων των εξυπηρετητών στο σύστημα διαχωριζόμενα με κόμμα.
system.ttp.id	Ορίζει το id του αξιόπιστου τρίτου μέρους.
system.numnettyworkers	Ορίζει τον αριθμό των netty νημάτων που δημιουργούνται σε κάθε εξυπηρετητή.
system.bft	Ορίζει κατά πόσο το σύστημα θα είναι ανθεκτικό σε σφάλματα κατάρρευσης η σε Βυζαντινά σφάλματα (true για Βυζαντινά σφάλματα και false

	για σφάλματα κατάρρευσης)
`system.samebatchsize`	Ορίζει κατά πόσο όλοι οι εξυπηρετητές θα στέλνουν τον ίδιο αριθμό από αιτήματα σε κάθε δέσμη στην εφαρμογή.

Πίνακας 4.1: Παράμετροι Διαμόρφωσης [10]

** Με κίτρινο χρώμα όσες παράμετροι χρειάζεται να μεταβάλουμε για την ΑΔΕ, στα υπόλοιπα διατηρείται η προεπιλεγμένη τιμή που είχαν.

Κεφάλαιο 5

Υλοποίηση Αλγορίθμων

5.1 Εισαγωγή

Στο κεφάλαιο αυτό θα εξηγήσουμε την υλοποίηση που δημιουργήσαμε για τους αλγόριθμους που αναλύθηκαν στο Κεφάλαιο 3 και τη μορφή των μηνυμάτων. Όπως προαναφέρθηκε χρησιμοποιούμε την βιβλιοθήκη BFT-Smart, οπότε χρειάζεται να εξηγήσουμε τις συναρτήσεις που χρησιμοποιήθηκαν.

5.2 Λειτουργίες Συστήματος BFT-Smart

Εδώ θα αναλύσουμε πως χρησιμοποιήσαμε το BFT-Smart για να υλοποιήσουμε τους αλγόριθμους του Κεφαλαίου 3. Πληροφορίες για το BFT-Smart υπάρχουν στο Κεφάλαιο 4 αλλά και στην βιβλιογραφία [4].

Αρχικά το BFT-Smart παρέχει πολλές δυνατότητες οι οποίες έκαναν την δουλειά μας πιο εύκολη, αφού εκτός ότι παρείχε demo που συνέβαλαν στην πλήρη

κατανόηση της βιβλιοθήκης, είναι υλοποιημένο για δημιουργία κατανεμημένων συστημάτων πελάτη-με σύνολο από εξυπηρετητές.
Η βιβλιοθήκη βασίζεται σε μεγάλο βαθμό στα 2 αρχεία διαμόρφωσής, το system.config και το host.config. Οι πιο πολλές παράμετροι στο system.config είναι αυτονόητες, παρόλα αυτά ο προγραμματιστής θα πρέπει να γνωρίζει εκ των προτέρων:

1. Τον αριθμό των εξυπηρετητών που θα αποτελούν το σύστημα (ορίζεται αυτός ο αριθμός στην παράμετρο διαμόρφωσης system.servers.num και είναι αναγκαίο να ορισθεί εκ των προτέρων για να μπορέσει το BFT-Smart να δημιουργήσει την όψη (view)).
2. Τον αριθμό των faulty εξυπηρετητών (system.servers.f).
3. Τι είδος σφάλματα ανέχεται το σύστημα (σφάλματα κατάρρευσης~ syste.bft=false η Βυζαντινά σφάλματα ~ system.bft=true)
4. Την IP διεύθυνση κάθε εξυπηρετητή και να την δηλώσει στην παράμετρο system.communication.bindaddress.

Ακολούθως στο αρχείο διαμόρφωσης host.config ο προγραμματιστής πρέπει να δηλώσει τις IP διευθύνσεις των εξυπηρετητών που αποτελούν το σύστημα, ακριβώς όπως εξηγήθηκε στο Κεφάλαιο 4. Το BFT-Smart παρέχει υλοποιημένη την συνάρτηση επικοινωνίας μεταξύ πελάτη και εξυπηρετητών, παρόλα αυτά η συνάρτηση πρέπει να γνωρίζει που να στείλει τα μηνύματα του πελάτη. Για αυτό τον λόγο ορίζονται εκ των προτέρων οι διευθύνσεις IP στο αρχείο hosts.config.

Μετά, αφού ορισθούν οι παράμετροι στα δύο αρχεία μπορεί να ξεκινήσει και η διαδικασία κλήσης των κλάσεων με χρήση των scripts. Τα scripts είναι γραμμένα από τους δημιουργούς της βιβλιοθήκης. Η κλήση των κλάσεων γίνεται με τους εξής τρόπους για Linux. (Στο Παράρτημα Ε εξηγείται πως γίνετε σε Windows καθώς και η ακριβής διαδικασία για χρήση του συστήματος).

Κλήση Πελάτη

./runscripts/smartrun.sh bftsmart/demo/blockchain/ClientGui **id πελάτη**

Κλήση Εξυπηρετητή

./runscripts/smartrun.sh bftsmart/demo/blockchain/BServer **id εξυπηρετητή IP διεύθυνση διαχειριστή**

Κλήση Faulty Εξυπηρετητή

./runscripts/smartrun.sh bftsmart/demo/blockchain/BFServer **id εξυπηρετητή IP διεύθυνση διαχειριστή**

Κλήση κλάσης που γίνεται επικοινωνία μέσω υποδοχής (Socket) με τους εξυπηρετητές για την δημιουργία log αρχείων που διαβάζει ο διαχειριστής

./runscripts/smartrun.sh bftsmart/demo/blockchain/SingleThreadedTCPServer **id εξυπηρετητή**

Κλήση διαχειριστή

./runscripts/smartrun.sh bftsmart/demo/blockchain/ServerAdministrator5Server **αριθμός εξυπηρετητών**

Η υλοποίηση του πελάτη σπάει σε δύο κλάσεις, στην κλάση ClientGui που είναι η γραφική του διαπροσωπεία και στην BClient που πραγματοποιεί την επικοινωνία με τους εξυπηρετητές. Στην BClient γίνεται εισαγωγή του navigators.smart.tom.ServiceProxy, το οποίο περιέχει τις μεθόδους για αποστολή μηνυμάτων από τον πελάτη στους εξυπηρετητές και την λήψη απαντήσεων από αυτούς (πιο συγκεκριμένα γίνεται χρήση της συνάρτησης invokeOrdered(byte[] request)).

Η συνάρτηση invokeOrdered είναι υλοποιημένη στην βιβλιοθήκη BFT-Smart και παρέχει ένα σύνολο από εγγυήσεις όπως την αποστολή μηνυμάτων σε όλους τους εξυπηρετητές, την επιστροφή της απαρτίας (τουλάχιστον $2f+1$ στην περίπτωση μας) ιδίων απαντήσεων από αυτούς και ότι τα αιτήματα φθάνουν στους εξυπηρετητές με βάση την χρονική σειρά που έγιναν (διάταξη πραγματικής ώρας).

Περισσότερες λεπτομέρειες σχετικά με τον προγραμματισμό της διεργασίας πελάτη θα δοθούν στο Κεφάλαιο 5.5.

Κατά την υλοποίηση του εξυπηρετητή γίνεται εισαγωγή του navigators.smart.tom.ServiceReplica το οποίο περιέχει συναρτήσεις για την λήψη αιτημάτων. Στην κλάση του εξυπηρετητή γίνεται λήψη αιτημάτων, επεξεργασία και αποστολή των απαντήσεων στην συνάρτηση invokeOrdered, όπου και γίνεται η καταμέτρηση των reply.

Πιο συγκεκριμένα όταν ο πελάτης καλέσει την invokeOrdered αυτή στέλνει το μήνυμα σε όλους τους εξυπηρετητές (το μήνυμα από πλευράς πελάτη μετατρέπεται σε byteStream περνιέται σαν παράμετρος στην invokeOrdered). Οι εξυπηρετητές το επεξεργάζονται και στέλνουν την απάντηση τους. Η πιο πάνω διαδικασία επιτυγχάνεται μέσω της συνάρτησης appExecuteOrdered. Περισσότερες λεπτομέρειες σχετικά με τον προγραμματισμό της διεργασίας πελάτη θα δοθούν στο Κεφάλαιο 5.4.

Οι υπόλοιπες κλάσεις του συστήματος δεν χρησιμοποιούν κάποια λειτουργία της βιβλιοθήκης BFT-Smart και θα εξηγηθούν στην συνέχεια του Κεφαλαίου 5.

5.3 Μορφή Μηνυμάτων

Όπως είπαμε και στα προηγούμενα κεφάλαια, ένα Κατανεμημένο Σύστημα είναι ένα σύστημα που αποτελείται από πολλές μηχανές, οι οποίες είτε είναι σε διαφορετικές περιοχές είτε είναι τοπικές. Στην περίπτωση που είναι σε διαφορετικές περιοχές, ο μόνος τρόπος επικοινωνίας μεταξύ τους είναι μέσω μηνυμάτων. Έτσι θα αναλύσουμε κάθε μορφής μήνυμα που μπορεί να προκύψει κατά την εκτέλεση του αλγορίθμου.

Αρχικά κάθε μήνυμα θα πρέπει να έχει κάποιο αναγνωριστικό έτσι ώστε να διαχωρίζεται από τα υπόλοιπα μηνύματα. Ο κάθε πελάτης έχει μια ταυτότητα μοναδική η οποία χρησιμοποιείται και στα μηνύματα, ο σκοπός είναι να αξιοποιήσουμε την μοναδικότητα της ταυτότητας πελάτη για να δημιουργήσουμε την ταυτότητα μηνύματος. Η ταυτότητα των μηνυμάτων γίνεται ως εξής:

Κάθε πελάτης έχει ένα αριθμό *c* ο οποίος σε κάθε αίτημα που εκτελεί αυξάνεται κατά ένα. Κάθε αίτημα πρέπει να ξεκινά με τον αριθμό των αιτημάτων που πραγματοποίησε ο πελάτης (*c*) πολλαπλασιασμένο με μια μεταβλητή *multi* (με αρχική τιμή 10) και αθροισμένο με την ταυτότητα του πελάτη (δηλαδή εκτελείται η πράξη *ClientID+multi*c*). Αυτό γίνεται μέχρι ο πελάτης να εκδώσει 9 αιτήματα. Μόλις

εκδώσει 9 αιτήματα η *c* γίνεται 1 και το multi του πελάτη πολλαπλασιάζεται επί 10. Έτσι τα αναγνωριστηκά μηνυμάτων του πελάτη με id = 1 θα είναι ως εξής 1 11 21 31…91..101 111 121 …

Τα μηνύματα πελάτη προς τους εξυπηρετητές μπορούν να έχουν δύο μορφές. Η πρώτη περίπτωση είναι της μορφής *(key,GET),* όπου key είναι το μοναδικό αναγνωριστικό του μηνύματος και GET (λήψη) είναι η ενέργεια που θέλουμε να πραγματοποιήσουν οι εξυπηρετητές. Σε αυτή την περίπτωση οι εξυπηρετητές επιστρέφουν την ακολουθία εγγραφών. Η άλλη μορφή μηνύματος είναι της μορφής *(key, APPEND,r),* όπου key είναι το μοναδικό αναγνωριστικό μηνύματος, APPEND η ενέργεια που θέλουμε να πραγματοποιήσουν οι εξυπηρετητές που στην περίπτωση μας είναι η πρόσθεση της εγγραφής r στην ακολουθία τους, και r η εγγραφή. Τα μηνύματα που στέλνουν οι εξυπηρετητές μεταξύ τους είναι ακριβώς τα ίδια με τα προαναφερθέντα.

Τα μηνύματα που στέλνουν οι εξυπηρετητές σαν ανταποκρίσεις στους πελάτες μπορούν να έχουν δύο μορφές. Στην περίπτωση που προηγουμένως εκδόθηκε ένα αίτημα λήψης τότε η απάντηση του εξυπηρετητή είναι της μορφής (S) όπου S είναι η ακολουθία του εξυπηρετητή, ακολούθως συγκρίνονται όλες οι απαντήσεις μεταξύ εξυπηρετητών και επιστρέφεται η απάντηση που επιστράφηκε από την απαρτία των εξυπηρετητών. Όσο αφορά τα αιτήματα πρόσθεσης, η απάντηση είναι της μορφής (ACK) όπου ACK είναι μια ένδειξη ότι έγινε η πρόσθεση με επιτυχία.

5.4 Μηχανισμός Ανοχής Βυζαντινών Σφαλμάτων

Το σύστημα είναι ανθεκτικό σε Βυζαντινές συμπεριφορές, όταν ικανοποιείται ο περιορισμός $n>=3f+1$ (*n* είναι ο συνολικός αριθμός εξυπηρετητών και *f* ο αριθμός βυζαντινών εξυπηρετητών). Στο σύστημα κάποιοι εξυπηρετητές έχουν Βυζαντινή συμπεριφορά. Βυζαντινή συμπεριφορά ορίστηκε ως η πρόσθεση λανθασμένης τιμής στην ακολουθία εγγράφων από τους Βυζαντινούς εξυπηρετητές, κάθε φορά που ο πελάτης αιτείται μια λειτουργία πρόσθεσης. Πιο συγκεκριμένα γίνεται πρόσθεση της εγγραφής “I’m faulty”.

5.5 Προγραμματισμός Διεργασιών Εξυπηρετητή

Κατά την υλοποίηση δημιουργήσαμε δύο κλάσεις για τους εξυπηρετητές. Η μια αντιπροσωπεύει τους Βυζαντινούς εξυπηρετητές και η άλλη τους εξυπηρετητές με την προβλεπόμενη συμπεριφορά. Το όνομα της κλάσης για τους Βυζαντινούς εξυπηρετητές είναι BFServer και για τους εξυπηρετητές που πραγματοποιούν την προβλεπόμενη συμπεριφορά είναι BServer. Οι δύο κλάσεις έχουν ακριβώς την ίδια υλοποίηση με την διαφορά ότι οι Βυζαντινοί εξυπηρετητές σε κάθε πρόσθεση, προσθέτουν την εγγραφή "Im'faulty". Κάθε εξυπηρετητής έχει τη δική του ταυτότητα, η οποία δηλώνεται κατά την εκκίνηση της κλάσης. Κατά την εκκίνηση πρέπει να δηλωθεί η IP διεύθυνση του διαχειριστή και η ταυτότητα του εξυπηρετητή. Οι πιο πάνω κλάσεις πέρα από τις λειτουργίες του εξυπηρετητή παρέχουν και γραφική διαπροσωπεία.

Με το που εκκινάται ένας εξυπηρετητής αυτόματα δημιουργείται ένα αντικείμενο της κλάσης BServer. Το αντικείμενο αρχικοποιεί το κατάστιχο (λίστα), δημιουργεί το log αρχείο το οποίο αποθηκεύει την ακολουθία του εξυπηρετητή, δημιουργεί ένα αντικείμενο τύπου Logger για την αποθήκευση log μηνυμάτων, και ένα αντικείμενο τύπου ServiceReplica, το οποίο παρέχει τις συναρτήσεις για λήψη μηνυμάτων από άλλους εξυπηρετητές και πελάτες.

Οι κυρίες συναρτήσεις που υλοποιήθηκαν ήταν η getSnapshot, η installSnapshot, η AppExecuteOrdered (υλοποιήθηκε και η βοηθητική συνάρτηση getSet) και η AppExecuteUnorderd.

Η συνάρτηση getSnapshot καλείται αυτόματα από την βιβλιοθήκη BFT-Smart ανά τακτά χρονικά διαστήματα και αποθηκεύει την κατάσταση του εξυπηρετητή σε μια ροή εξόδου. Αυτό γίνεται σε διαφορετικούς εξυπηρετητές κάθε φορά και αποσκοπεί στην παροχή της πιο πρόσφατης κατάστασής σε κάποιον εξυπηρετητή ο οποίος δεν την έχει. Η ενημέρωση της κατάστασης του εξυπηρετητή γίνεται μέσω της installSnapshot η οποία αντιγράφει το αντικείμενο κατάστιχου από την ροή εισόδου στην ακολουθία εγγραφών του συγκεκριμένου εξυπηρετητή.

```
        @Override
        public byte[] getSnapshot() {
                try (ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
                                ObjectOutput objOut = new ObjectOutputStream(byteOut)) {
                        objOut.writeObject(LedgerFinal);
                        return byteOut.toByteArray();
                } catch (IOException e) {
                        logger.log(Level.SEVERE, "Error while taking snapshot", e);
                }
                return new byte[0];
        }

        @SuppressWarnings("unchecked")
        @Override
        public void installSnapshot(byte[] state) {
                try (ByteArrayInputStream byteIn = new ByteArrayInputStream(state);
                                ObjectInput objIn = new ObjectInputStream(byteIn)) {
                        LedgerFinal = (List<String>)objIn.readObject();
                } catch (IOException | ClassNotFoundException e) {
                        logger.log(Level.SEVERE, "Error while installing snapshot", e);
                }
        }
}
```

Σχημα 5.1: Κωδικας συνάρτησης getSnapshot και installSnapshot

Ακολούθως έχουμε την AppExecuteOrdered και AppExecuteUnordered που αποτελούν τον πυρήνα κάθε εξυπηρετητή. Τόσο η μια συνάρτηση όσο και η άλλη είναι υπεύθυνες για να λαμβάνουν αιτήματα από τους πελάτες, να τα επεξεργάζονται και να στέλνουν την ανταπόκριση πίσω σε αυτούς. Η διαφορά μεταξύ των δυο συναρτήσεων είναι πως η AppExecuteOrdered είναι για αιτήματα για γράψιμο (πρόσθεση) και διάβασμα (λήψη) ενώ η AppExecuteUnorderd είναι μόνο για αιτήματα για διάβασμα (λήψη). Η AppExecuteOrdered επεξεργάζεται τα αιτήματα με την σειρά πραγματικής ώρας ενώ η AppExecuteUnordered όχι. Στην υλοποίηση μας χρησιμοποιείται μόνο η AppExecuteOrdered και η AppExecuteUnordered υλοποιήθηκε σαν μια επιπλέον λειτουργία.

Κατά την εκτέλεση της AppExecuteOrdered αρχικοποιείται ένας πίνακας από bytes με όνομα reply (εκεί αποθηκεύεται η απάντηση που θα στείλει ο εξυπηρετητής στους πελάτες), μια μεταβλητή key (αποθηκεύει το αναγνωριστικό αιτήματος), μια μεταβλητή value που χρησιμοποιείται σε περίπτωση λειτουργίας πρόσθεσης (περιέχει την τιμή της εγγραφής που θα προστεθεί) και μια τιμή Boolean με όνομα hasReply (παίρνει τιμή true αν υπάρχει reply διαφορετικά παίρνει τιμή false).

Επίσης αρχικοποιείται μια ροή εισόδου με όνομα objIn (ObjectInput objIn) και ένας πίνακας από byte με όνομα byteIn (ByteArrayInputStream byteIn). Η ροή εισόδου objIn χρησιμοποιείται για παραλαβή αιτημάτων από τους πελάτες. Ο πίνακας από byte, byteIn χρησιμοποιείται για αποθήκευση δεδομένων που λήφθηκαν από την ροή

εισόδου, για να αξιοποιηθούν από το πρόγραμμα εξυπηρετητή. Τέτοια δεδομένα είναι το είδος αιτήματος, το αναγνωριστικό του, και η τιμή εγγραφής.

Επίσης αρχικοποιείται μια ροή εξόδου με όνομα objOut (ObjectOutput objOut) και ένα πίνακα από bytes με όνομα byteOut (ByteArrayOutputStream). Η ροή εξόδου χρησιμοποιείται για να το γράψιμο δεδομένων που θα ληφθούν σαν είσοδο από κάποια άλλη διεργασία, στην προκειμένη περίπτωση η απάντηση που λαμβάνουν οι πελάτες.

Πιο αναλυτικά στην συνάρτηση λαμβάνουμε από την ροή εισόδου τρεις πληροφορίες, το είδος του αιτήματος, το αναγνωριστικό του αιτήματος και αν το αίτημα είναι αίτημα πρόσθεσης την τιμή που θα προστεθεί. Αν το αίτημα είναι αίτημα πρόσθεσης τοποθετούμε την τιμή στην τελευταία θέση στην ακολουθία εγγραφών, γράφουμε την νέα τιμή στο αρχείο log και στέλνουμε μέσω υποδοχής την νέα τιμή που προσαρτήθηκε στην κλάση SingleThreadedTCPServer. (δημιουργεί το αρχείο από το οποίο διαβάζει ο διαχειριστής για να παρουσιάσει τα περιεχόμενα κάθε εξυπηρετητή). Ακολούθως στέλνετε ACK στον πελάτη (γράφεται στην ροή εξόδου objOut).

Σε περίπτωση που το αίτημα είναι αίτημα λήψης καλείται η βοηθητική συνάρτηση getSet, που γράφει στην ροή εξόδου την ακολουθία εγγραφών (τα περιεχόμενα της ροής εξόδου είναι η απάντηση που λαμβάνουν οι πελάτες). Τέλος στέλνεται η απάντηση στον πελάτη. Υπάρχει ολόκληρος ο κώδικας στο Παράρτημα Α.
(Επειδή για κάθε εξυπηρετητή παρέχεται γραφική διπροσωπία, κάθε πέντε δευτερόλεπτα ελέγχεται αν υπήρξε κάποια πρόσθεση, σε περίπτωση που υπήρξε τότε ενημερώνεται η γραφική διαπροσωπεία με τις πιο πρόσφατες τιμές του αντικείμενου κατάστιχου).

```
292         public byte[] appExecuteOrdered(byte[] command, MessageContext msgCtx) {
293                 byte[] reply = null;
294                 K key = null;
295                 V value = null;
296                 boolean hasReply = false;
297                 System.out.print("The ledger has the following records :");
298
299                 for (int i=0; i<LedgerFinal.size(); i=i+1){
300                         System.out.print(" "+LedgerFinal.get(i));
301                         }
302
303                 System.out.println();
304                 try (ByteArrayInputStream byteIn = new ByteArrayInputStream(command);
305                                 ObjectInput objIn = new ObjectInputStream(byteIn);
306                                 ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
307                                 ObjectOutput objOut = new ObjectOutputStream(byteOut);) {
308
309                         BRequestType reqType = (BRequestType)objIn.readObject();
310                         switch (reqType) {
311                                 case APPENDLEDGER:
312
313                                         key = (K)objIn.readObject();
314                                         System.out.println("The APPEND message with C="+key+" has been delivered");
315                                         value = (V)objIn.readObject();
316                                         String valuetemp=(String)value;
317                                         LedgerFinal.add(valuetemp);
318
319                                         System.out.println(LedgerFinal.size());
320                                         try {
321                                                 FileWriter myWriter = new FileWriter(myObj,true);
322                                                 String tmpValueWrite=(String)value;
323                                                 myWriter.append(tmpValueWrite+"\n");
324                                                 myWriter.close();
325                                                 System.out.println("Successfully wrote to the file.");
326                                                 int Portnum=8080+PortId;
327                                                 Socket socket = new Socket(IpAddAdmin, Portnum);
328                                                 DataOutputStream outputSock = new
DataOutputStream(socket.getOutputStream());
329                                                 outputSock.writeBytes(tmpValueWrite);
330                                                 socket.close();
331                                             }
332                                             catch (IOException e) {
333                                                 System.out.println("An error occurred.");
334                                                 e.printStackTrace();
335                                                 }
336
337
338                                         String Response1 = "ACK";
339                                         System.out.println("The "+Response1+" send to the client");
340                                         if (Response1 != null) {
341                                                 objOut.writeObject(Response1);
342                                                 hasReply = true;
343                                         }
344                                         break;
345                                 case GETLEDGER:
346
347                                         key = (K)objIn.readObject();
348                                         System.out.println("The GET message with C="+key+" has been delivered");
349                                         getset(objOut);
350                                         hasReply = true;
351
352
353                                         break;
354
355                         }
356                         if (hasReply) {
357                                 objOut.flush();
358                                 byteOut.flush();
359                                 reply = byteOut.toByteArray();
360                         } else {
361                                 reply = new byte[0];
362                         }
363
364                 } catch (IOException | ClassNotFoundException e) {
365                         logger.log(Level.SEVERE, "Ocurred during map operation execution", e);
366                 }
367                 return reply;
368         }
369
```

Σχημα 5.2: Κωδικας συνάρτησης appExecuteOrdered

5.5.1 Πρόγραμμα Διεργασιών Faulty Εξυπηρετητή

Ο Βυζαντινός εξυπηρετητής έχει ακριβώς τον ίδιο κώδικα με τον κώδικα του εξυπηρετητή με την προβλεπόμενη συμπεριφορά. Η διαφορά είναι πως σε κάθε πρόσθεση αντί στην ακολουθία εγγραφών να προστίθεται η εγγραφή που επιθυμεί ο πελάτης να προστίθεται η εγράφη “I am faulty ”. Το σημείο όπου διαφέρουν οι 2 κώδικες φαίνεται στο Σχήμα 5.3 και ολόκληρος ο κώδικας βρίσκεται στο Παράρτημα Β.

```
310                 key = (K)objIn.readObject();
311                 System.out.println("The APPEND message with C="+key+" has been delivered");
312                 value = (V)objIn.readObject();
313                 String valuetemp="Im faulty";
314                 LedgerFinal.add(valuetemp);
315
316                 System.out.println(LedgerFinal.size());
```

Σχήμα 5.3: Κώδικας που διαφέρει από BServer

5.5.2 Πρόγραμμα Επικοινωνίας Με Εξυπηρετητές

Η κλάση SingleThreadedTCPServer είναι υπεύθυνη για την επικοινωνία με τους εξυπηρετητές. Κάθε εξυπηρετητής επικοινωνεί με ένα διαφορετικό στιγμιότυπο της κλάσης SingleThreadedTCPServer (κάθε εξυπηρετητής επικοινωνεί με το στιγμιότυπο του προγράμματος επικοινωνίας που έχει το ίδιο αναγνωριστικό με αυτόν) και του στέλνει τις εγγραφές που προσθέτει στην ακολουθία εγγραφών του.

Συγκεκριμένα σε αυτήν την κλάση δημιουργείται μια υποδοχή η οποία ακούει σε διαφορετική πόρτα για κάθε εξυπηρετητή (οι εξυπηρετητές συμπεριφέρονται σαν πελάτες οι οποίοι κάθε φορά που πραγματοποιείται μια λειτουργία πρόσθεσης αυτοί συνδέονται με την αντίστοιχη κλάση SingleThreadedTCPServer και της στέλνουν την νέα τιμή που προσαρτήθηκε). Ακολούθως ακούει συνεχόμενα για αιτήσεις, (μέσω της μεθόδου accept), μόλις η μέθοδος accept ξεμπλοκαριστεί τότε πραγματοποιήθηκε μια λειτουργία πρόσθεσης και ο εξυπηρετητής έστειλε την νέα τιμή που προστέθηκε. Αυτή η τιμή προστίθεται στην λίστα του στιγμιότυπου (η οποία είναι μοναδική για κάθε στιγμιότυπο και συνάμα για κάθε εξυπηρετητή) και στην αρχή του αρχείου LogFileServerId.txt (όπου το ServersId είναι η μοναδική ταυτότητα του εξυπηρετητή). Η πιο πάνω διαδικασία πραγματοποιείται επ'αόριστον μέχρι να τερματιστεί η λειτουργία του προγράμματος. Υπάρχει ολόκληρος ο κώδικας στο Παράρτημα Γ.

5.5.3 Πρόγραμμα Παρουσίασης Περιεχομένων Όλων των Εξυπηρετητών

Ο κύριος σκοπός της κλάσης αυτής (ServerAdministrator5Server) είναι καθαρά για γραφική διαπροσωπεία. Πιο συγκεκριμένα τρέχει στον ίδιο υπολογιστή που τρέχουν και οι κλάσεις SingleThreadTCPServer, διαβάζει από τα αρχεία LogFileServerId.txt (ένα για κάθε εξυπηρετητή) και παρουσιάζει τα περιεχόμενα τους στην οθόνη υπό

μορφή γραφικής διαπροσωπείας. Μέσω αυτής της κλάσης ο διαχειριστής του συστήματος μπορεί να βλέπει τα περιεχόμενα κάθε εξυπηρετητή ανά πάσα στιγμή. Η μόνη παράμετρος που παίρνει η κλάση είναι ο αριθμός των εξυπηρετητών, και η λειτουργία της βασίζεται στην συνάρτηση timer αφού κάθε 5 δευτερόλεπτα ελέγχεται αν άλλαξε κάτι στα αρχεία LogFileServerId.txt, αν άλλαξε κάτι τα περιεχόμενα της γραφικής διαπροσωπείας ενημερώνονται (υπάρχει ενημέρωση κάθε 5 δευτερόλεπτα).

5.6 Προγραμματισμός Διεργασιών Πελάτη

Αρχικά η υλοποίηση του πελάτη χωρίζεται σε 3 κλάσεις, την BRequestType που περιέχει ένα απλό enum με τις λειτουργίες του Αντικειμένου Κατάστιχου (λήψη και πρόσθεση). Την κλάση ClientGui που είναι γραφική διαπροσωπεία του πελάτη και η κλάση BClient που υλοποιεί τις λειτουργίες πελάτη.

Στην κλάση ClientGui δημιουργείται το αντικείμενο BClient και η γραφική διαπροσωπεία για τον πελάτη. Η γραφική διαπροσωπεία έχει 2 κουμπιά, το πρώτο είναι υπεύθυνο για την λειτουργία πρόσθεσης και το δεύτερο για την λειτουργία λήψης. Όταν πατηθεί το κουμπί πρόσθεσης, υπολογίζεται το αναγνωριστικό μηνύματος και καλείται η συνάρτηση (στην γραφική διαπροσωπεία δεν υπάρχει η έννοια της συνάρτησης άλλα του Action Listener που όταν πατηθεί κάποιο κουμπί εκτελείται ένας συγκεκριμένος κώδικας, μέσα σε αυτόν τον κώδικα γίνεται η κλήση της συνάρτησης που στέλνει το αίτημα πρόσθεσης στους εξυπηρετητές την AppendLedger) που είναι υπεύθυνη για την λειτουργία πρόσθεσης μέσω του αντικειμένου πελάτη. Ακολούθως ο πελάτης ενημερώνεται μέσω της γραφικής διαπροσωπείας, κατά πόσο το μήνυμα στάλθηκε (broadcast) στους εξυπηρετητές και κατά πόσο λήφθηκε η απάντηση από τουλάχιστον f+1 διαφορετικούς εξυπηρετητές (αυτό επιτυγχάνεται με την χρήση ετικετών που είναι τοποθετημένες στο πλαίσιο). Το δεύτερο κουμπί είναι υπεύθυνο για την λειτουργία της λήψης, όταν πατηθεί καλείται η συνάρτηση (στην γραφική διαπροσωπεία δεν υπάρχει η έννοια της συνάρτησης άλλα του Action Listener όπου όταν πατηθεί κάποιο κουμπί εκτελείται ένας συγκεκριμένος κώδικάς ο οποίος όμως περιέχει την κλήση της συνάρτησης GetLedger) που είναι υπεύθυνη για την λειτουργία της λήψης και αυτό επιτυγχάνεται με την χρήση του αντικειμένου BClient, στην συνέχεια ενημερώνεται ο πελάτης μέσω της γραφικής διαπροσωπείας κατά πόσο το μήνυμα στάλθηκε (broadcast) στους εξυπηρετητές και κατά πόσο λήφθηκε η απάντηση από f+1 διαφορετικούς

εξυπηρετητές (αυτό γίνεται μέσω ετικετών που είναι τοποθετημένες στο πλαίσιο). Τέλος ο πελάτης είναι σε θέση να δει τα περιεχόμενα του αντικείμενου κατάστιχου (στην περίπτωση που υπάρχουν πάνω από 14 εγγραφές υπάρχει η επιλογή να μεταβεί στην επόμενη σελίδα για να δει και τις υπόλοιπες εγγραφές άλλα και να επιστρέψει σε κάποια προηγούμενη σελίδα).

Τέλος η κλάση BClient είναι η κύρια διεργασία πελάτη, η οποία χρειάζεται για να στείλουμε αιτήματα στους εξυπηρετητές, είτε για πρόσθεση εγγραφών στο αντικείμενο κατάστιχου είτε για λήψη του αντικείμενου κατάστιχου (λίστα FinalLedger). Η διεργασία πελάτη έχει 2 συναρτήσεις την AppendLedger και την GetLedger. Στην AppendLedger, χρησιμοποιείται μια ροή εξόδου (ObjectOutput objOut) και ένας πίνακας από bytes (ByteArrayOutputStream byteOut) που αποθηκεύει τα δεδομένα που θα φύγουν σαν έξοδος από την διεργασία πελάτη μέσω της ροής εξόδου. Αυτά τα δεδομένα είναι η μορφή του αιτήματος (αίτημα πρόσθεσης), το αναγνωριστικό αιτήματος και τα δεδομένα εγγραφής. Ακολούθως καλείται η συνάρτηση serviceProxy.invokeOrdered η οποία στέλνει της πληροφορίες που είναι αποθηκευμένες στο byteOut σε όλους τους εξυπηρετητές και επιστρέφει την απάντηση τους (ACK). Για να διαβαστεί η απάντηση επειδή είναι πίνακας από byte απαιτείται μια ροή εισόδου (ObjectInput objIn) και ένας πίνακας από bytes (ByteArrayInputStream byteIn). Υπάρχει ολόκληρος ο κώδικάς στο παράρτημα Δ.

```
29        public V AppendLedger(K key, V value) {
30                try (ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
31                                ObjectOutput objOut = new ObjectOutputStream(byteOut);) {
32
33                        objOut.writeObject(BRequestType.APPENDLEDGER);
34                        objOut.writeObject(key);
35                        objOut.writeObject(value);
36
37                        objOut.flush();
38                        byteOut.flush();
39                        System.out.println("The message APPEND with C="+key+" has been broadcasted to the servers of
  the Ledger");
40
41                        byte[] reply = serviceProxy.invokeOrdered(byteOut.toByteArray());
42                        System.out.println("The same reply from at least f+1 different servers for the "+key+"
  request have just arrived");
43                        if (reply.length == 0)
44                                return null;
45                        try (ByteArrayInputStream byteIn = new ByteArrayInputStream(reply);
46                                        ObjectInput objIn = new ObjectInputStream(byteIn)) {
47
48                                return (V)objIn.readObject();
49                        }
50
51                } catch (IOException | ClassNotFoundException e) {
52                        System.out.println("Exception putting value into map: " + e.getMessage());
53                }
54                return null;
55        }
56
```

Σχημα 5.4: Συνάρτηση AppendLedger

Στην περίπτωση της λήψης εκτελείται η GetLedger. Γίνεται χρήση μιας ροή εξόδου (ObjectOutput objOut) και ενός πίνακα από bytes (ByteArrayOutputStream byteOut) που αποθηκεύει τα δεδομένα που θα φύγουν σαν έξοδος από την διεργασία πελάτη μέσω της ροής εξόδου. Στην περίπτωση της λήψης αυτά τα δεδομένα είναι η μορφή του αιτήματος (αίτημα λήψης), και το αναγνωριστικό αιτήματος. Ακολούθως καλείται η συνάρτηση serviceProxy.invokeOrdered η οποία στέλνει τις πληροφορίες που είναι αποθηκευμένες στο byteOut σε όλους τους εξυπηρετητές και επιστρέφει την απάντηση τους. Στην περίπτωση λήψης είναι το αντικείμενο κατάστιχου. Για να διαβαστεί η απάντηση επειδή είναι πίνακας από byte απαιτείται μια ροή εισόδου (ObjectInput objIn) και ένας πίνακας από bytes (ByteArrayInputStream byteIn).

```
57        @SuppressWarnings("unchecked")
58        public List<String> GetLedger(Object key) {
59                try (ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
60                                ObjectOutput objOut = new ObjectOutputStream(byteOut);) {
61
62                        objOut.writeObject(BRequestType.GETLEDGER);
63                        objOut.writeObject(key);
64
65                        objOut.flush();
66                        byteOut.flush();
67                        System.out.println("The message GET with C="+key+" has been broadcasted to the servers of
   the Ledger");
68
69                        byte[] reply = serviceProxy.invokeOrdered(byteOut.toByteArray());
70                        try (ByteArrayInputStream byteIn = new ByteArrayInputStream(reply);
71                                        ObjectInput objIn = new ObjectInputStream(byteIn)) {
72                                System.out.println("The same reply from at least f+1 different servers for the "+key+"
   request have just arrived");
73                                int size = objIn.readInt();
74                                List<String> result = new ArrayList<String>();
75                                while (size-- > 0) {
76                                        result.add((String)objIn.readObject());
77                                }
78                                return result;
79                        }
80
81                } catch (IOException | ClassNotFoundException e) {
82                        System.out.println("Exception getting keyset from map: " + e.getMessage());
83                }
84                return null;
85        }
86
```

Σχημα 5.5: Συνάρτηση GetLedger

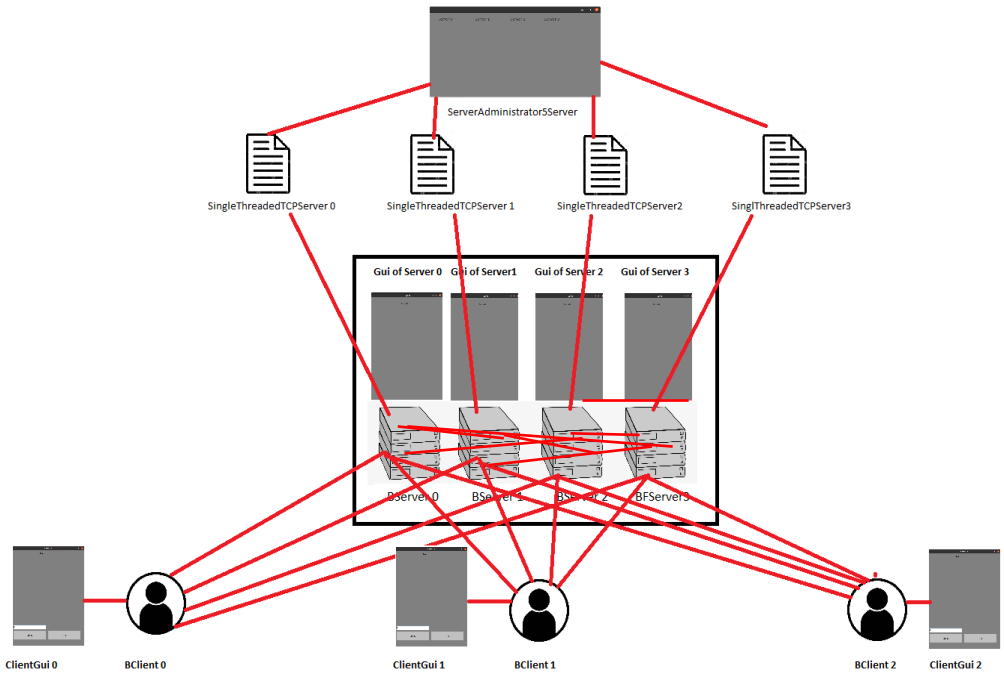
5.7 Ολοκληρωμένο Παράδειγμα Εκτέλεσης

Σε ένα παράδειγμα εκτέλεσης η διαδρομή που ακολουθείται είναι η εξής. Αρχικά πρέπει να αρχικοποιηθούν όλοι οι εξυπηρετητές, αυτό επιτυγχάνεται με το τρέξιμο της κλάσης BServer σε διαφορετικές μηχανές (εάν επιθυμούμε ο εξυπηρετητής να έχει Βυζαντινή συμπεριφορά τότε τρέχουμε την κλάση BFServer). Ακολούθως στον υπολογιστή του διαχειριστή εκτελείται η κλάση SingleThreadedTCPServer μια φορά

για κάθε εξυπηρετητή, μετά εκτελείται η ServerAdministrator5Server. Αφού γίνει η πιο πάνω διαδικασία οι πελάτες είναι έτοιμοι να ξεκινήσουν τα αιτήματα.

Για να εισαχθεί ένας πελάτης στο σύστημα εκτελείται η κλάση ClientGui η οποία στο εσωτερικό της δημιουργεί ένα αντικείμενο πελάτη (BClient). Μέσω της ClientGui εμφανίζεται η γραφική διαπροσωπεία για τον πελάτη. Όταν ο πελάτης πραγματοποιεί μια αίτηση, το αίτημα μεταφέρεται στου εξυπηρετητές όπου και το επεξεργάζονται. Ακολούθως στέλνεται η ανταπόκρισή από τουλάχιστον f+1 εξυπηρετητές στον πελάτη.

Τέλος αν το αίτημα είναι αίτημα πρόσθεσης οι εξυπηρετητές στέλνουν την νέα εγγραφή στην κλάση SingleThreadedTCPServer η οποία την προσθέτει στο αρχείο από το οποίο διαβάζει η ServerAdministrator5Server για να τα παρουσιάσει μέσω της γραφικής διαπροσωπείας.



Σχήμα 5.6: Ολοκληρωμένο Παράδειγμα Εκτέλεσης

Κεφάλαιο 6

Διαπροσωπεία

6.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τις λειτουργίες που προσφέρει η κάθε διαπροσωπεία και θα παρουσιάσουμε τις εισόδους και τις εξόδους της. Στο τέλος θα αναφέρουμε περιορισμούς που εμφανίστηκαν κατά την υλοποίηση.

6.2 Περιγραφή

Στόχος της διαπροσωπείας είναι η εύκολη χρήση του συστήματος από τους χρήστες (πελάτες), και η παρουσίαση του αντικειμένου κατάστιχου μέσα από τους εξυπηρετητές και τον διαχειριστή. Συνολικά έχουμε 4 κλάσεις οι οποίες παράγουν γραφική διαπροσωπεία, η ClientGui η οποία δίνει την δυνατότητα στον πελάτη να πραγματοποιεί αιτήματα στους εξυπηρετητές και να δείχνει τα αποτελέσματα των αιτημάτων του όπως αυτά περιγράφονται στους αλγορίθμους του Κεφαλαίου 4. Την BServer (και BFServer) η οποία παρουσιάζει τα περιεχόμενα του αντικειμένου κατάστιχου για κάθε εξυπηρετητή και η ServerAdministrator5Server που παρουσιάζει τα περιεχόμενα όλων των εξυπηρετητών σε μια οθόνη. Για να αρχικοποιηθεί η κάθε διαπροσωπεία απλά τρέχουμε τις προαναφερθέν κλάσεις.

Το εργαλείο που χρησιμοποιήθηκε ήταν το WindowBuilder [9], ένα plugin για Java το οποίο χρησιμοποιήθηκε για να κάνουμε στο πλαίσιο (frame) drag and drop τα διάφορα widget (ετικέτες, κουμπιά κτλ.) και για να εκτελούνται οι απαραίτητες

λειτουργίες με το πάτημα κουμπιών (ορίστηκαν συναρτήσεις μέσα σε ActionListener που είναι συναρτήσεις όπου εκτελούνται όταν πατηθεί κάποιο κουμπί).

6.3 Παράθυρο Εξυπηρετητή

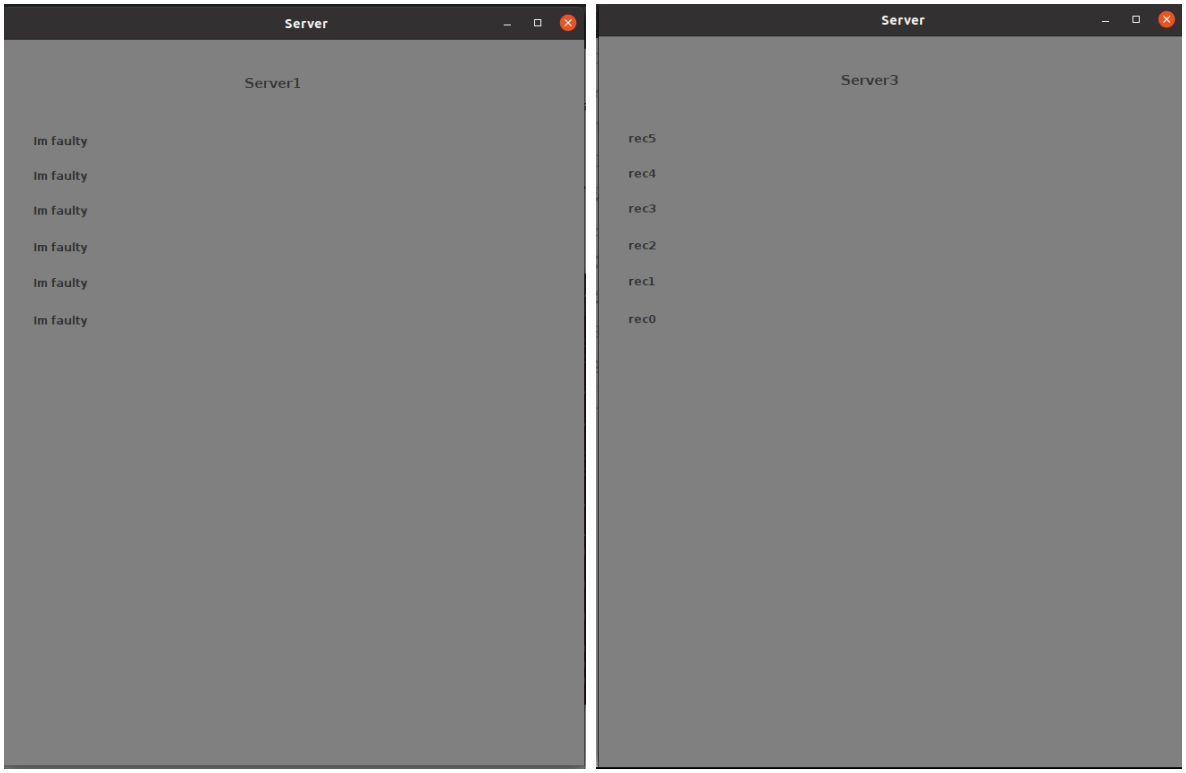
Η υλοποίηση του παραθύρου εξυπηρετητή είναι σχετικά απλή αφού αποτελείται από 16 ετικέτες, μια για τον τίτλο και 15 που παρουσιάζουν τις 15 πιο πρόσφατες εγγραφές.

Όλες οι ετικέτες είναι σε ένα πλαίσιο με απόλυτη διάταξη. Επιλέχτηκε απόλυτη διάταξη για να υπάρχει η ελευθέρια μετακίνησης των widget παντού στο πλαίσιο. Μέσω της εντολής αυτής frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); επιβάλλεται στο πλαίσιο να κλείσει όταν πατηθεί το κουμπί χ πάνω δεξιά.

Στην συνέχεια ορίστηκε που θέλουμε να βρίσκεται το πλαίσιο καθώς και το χρώμα του. Αυτό έγινε μέσω των εντολών frame.setBounds(100, 100, 649, 850); και (contentPane.setBackground(Color.GRAY);).

Ακολούθως προσθέσαμε τον τίτλο (βρίσκεται στην πρώτη γραμμή και είναι της μορφής Server #ID) και τις 15 ετικέτες οι οποίες όλες έχουν πλάτος 200, ύψος 16, ξεκινούν όλες από την θέση 33 στον άξονα χ ενώ η θέση τους στον άξονα ψ έχει διάφορα 40 από ετικέτα σε ετικέτα.

Τέλος ορίσαμε ένα timer το οποίο κάθε 5 δευτερόλεπτα ανανεώνει αυτές τις ετικέτες, σε περίπτωση που έγινε κάποια ενέργεια πρόσθεσης η πιο πρόσφατη εγγραφή τοποθετείται πρώτη και οι άλλες μεταφέρονται μια θέση κάτω από την προηγουμένη τους, σε περίπτωση που δεν έγινε εγγραφή τότε δεν πραγματοποιείται καμία αλλαγή.



Σχήμα 6.1: Γραφική διαπροσωπεία εξυπηρετητή και faulty εξυπηρετητή

6.4 Παράθυρο Διαχειριστή

Το παράθυρο για τον διαχειριστή είναι λίγο πιο περίπλοκο από αυτό του εξυπηρετητή μιας και πρέπει να παρουσιάζει τα περιεχόμενα όλων των εξυπηρετητών. Επίσης είναι δυναμικό αφού ο αριθμός των widget που παρουσιάζονται στην οθόνη εξαρτάται από την παράμετρο που θα δοθεί από την γραμμή εντολής, παρόλα αυτά τα widget είναι τα ίδια για κάθε εξυπηρετητή. Τέλος παρέχει 2 κουμπιά το ένα είναι το κουμπί next, που σε περίπτωση που ο εξυπηρετητής έχει πάνω από 5 εγγραφές δίνεται η δυνατότητα να πατήσουμε next και να δούμε τις εγγραφές που δεν χωράνε στην οθόνη (επόμενη σελίδα). Το δεύτερο κουμπί είναι το previous το οποίο μας παίρνει στην προηγουμένη σελίδα. Οι πιο πρόσφατες εγγραφές εμφανίζονται πρώτες.

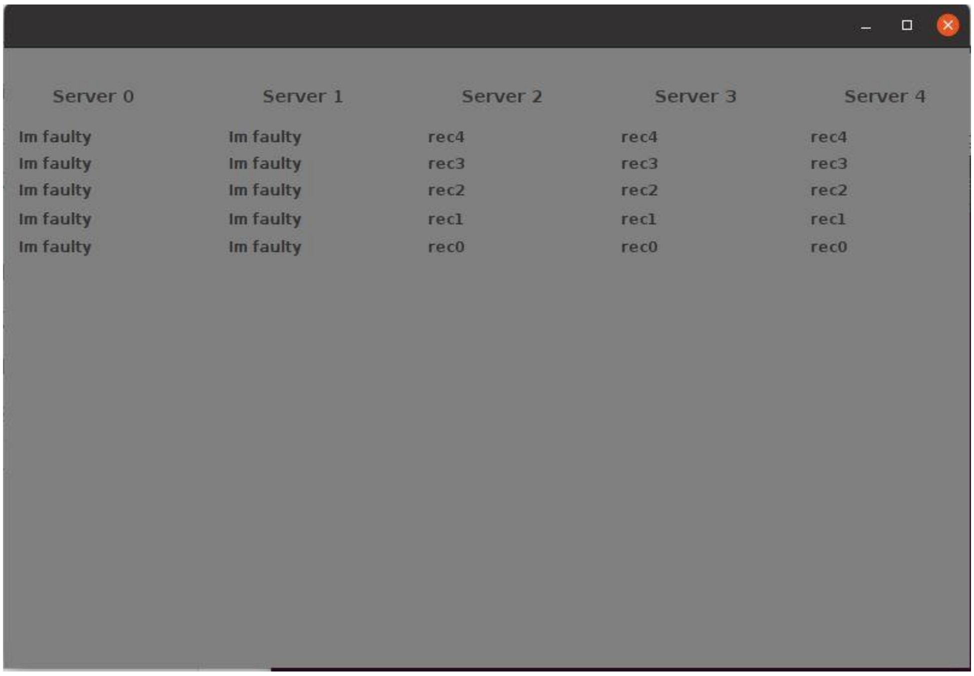
Θα περιγράψουμε τα περιεχόμενα του γραφικού περιβάλλοντος θεωρώντας ότι στο σύστημα υπάρχει μόνο ένας εξυπηρετητής μιας και θα ήταν πλεονασμός να περιγράψουμε για περισσοτέρους αφού για κάθε εξυπηρετητή επαναλαμβάνεται ουσιαστικά ο ίδιος κώδικας.

Πιο αναλυτικά όλα τα κουμπιά και οι ετικέτες είναι σε ένα πλαίσιο. Κατά την δημιουργία του παραθύρου ορίσαμε την διάταξη του σαν απόλυτη, έτσι ώστε να έχουμε την ελευθερία να μετακινούμε τα widget όπου θέλουμε. Μέσω της εντολής frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); επιβάλουμε στο πλαίσιο να κλείσει όταν πατηθεί το κουμπί χ πάνω δεξιά. Στην συνέχεια ορίσαμε που θέλουμε να βρίσκεται το πλαίσιο (frame.setBounds(100, 100, 800, 550);) και τι χρώμα να έχει (contentPane.setBackground(Color.GRAY);).

Ακολούθως προσθέσαμε τις 6 ετικέτες η μια δηλώνει τον τίτλο του εξυπηρετητή και οι άλλες 5 τις 5 πιο πρόσφατες εγγραφές, οι οποίες όλες έχουν πλάτος 100 και ύψος 20, ξεκινούν όλες από την θέση 12 στον άξονα χ ενώ η θέση τους στον άξονα ψ έχει διαφορά 20 από ετικέτα σε ετικέτα. Επίσης ορίσαμε ένα timer το οποίο κάθε 5 δευτερόλεπτα ανανεώνει αυτές τι ετικέτες, σε περίπτωση που έγινε κάποια ενέργεια πρόσθεσης η πιο πρόσφατη εγγραφή τοποθετείται πρώτη και οι άλλες μεταφέρονται μια θέση κάτω από την προηγουμένη τους. Σε περίπτωση που δεν έγινε κάποια πρόσθεση τότε δεν πραγματοποιείται καμία αλλαγή.

Για κάθε εξυπηρετητή ορίσαμε μια λίστα η οποία έχει τα περιεχόμενα του αντικειμένου κατάστιχου (σε αντιστροφή σειρά) αυτού του εξυπηρετητή, μέρος της συνάρτησης που καλείται κάθε 5 δευτερόλεπτά είναι να διαβάζει από το αρχείο Serveridlog.txt το οποίο παράγεται από την κλάση SingleThreadTCPServer όπως εξηγήθηκε στο Κεφαλαίο 5, και να αποθηκεύει τα περιεχόμενα του στην λίστα. Τα περιεχόμενα αυτής της λίστας εμφανίζονται στις ετικέτες (στην λίστα στην θέση 0 βρίσκεται η πιο πρόσφατη εγγραφή και στην θέση n η πιο παλιά).

Τέλος υπάρχει ένας μετρητής ο οποίος μετρά πόσες εγγραφές έχει μέσα η λίστα, σε περίπτωση που υπάρχουν πάνω από 5 εγγραφές τότε ενεργοποιείται το κουμπί next το οποίο μας εμφανίζει τις επόμενες 5 εγγραφές στην λίστα αυξάνοντας και τον μετρητή. Μόλις πατηθεί μια φορά το κουμπί next ενεργοποιείται το κουμπί previous που εμφανίζει τις προηγούμενες πέντε εγγραφές μειώνοντας και τον μετρητή.



Σχήμα 6.2: Παρουσίαση γραφικής διαπροσωπείας για διαχειριστή με λιγότερες από 5 εγγραφές

Σχήμα 6.3: Παρουσίαση γραφικής διαπροσωπείας για διαχειριστή με περισσότερες από 5 εγγραφές

Σχήμα 6.4 : Παρουσίαση γραφικής διαπροσωπείας για διαχειριστή με περισσότερες από 5 εγγραφές όπου πατήθηκε το κουμπί next

6.5 Παράθυρο Πελάτη

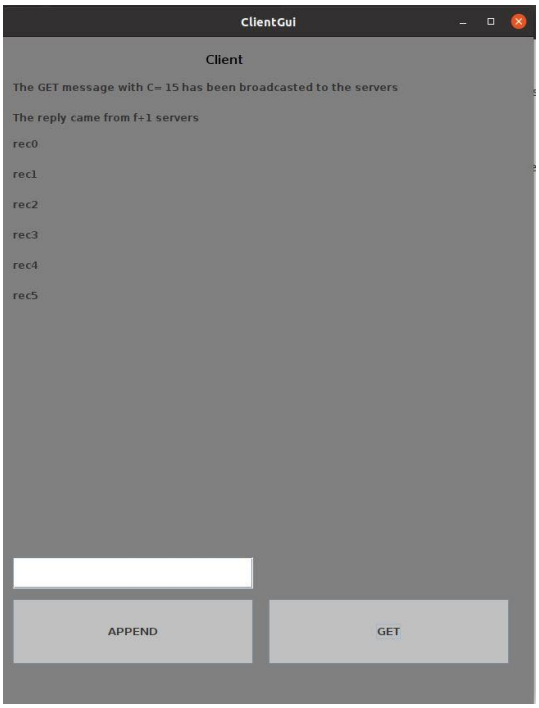
Το παράθυρο για τον πελάτη είναι αρκετά παρόμοιο με αυτό του διαχειριστή για την περίπτωση που παρουσιάζει δεδομένα μόνο για ένα εξυπηρετητή, η μόνη διαφορά είναι στον αριθμό εγγράφων αφού αντί 5 εγγραφές που παρουσιάζει ο διαχειριστής ο πελάτης παρουσιάζει 14.

Η γραφική διεπαφή του πελάτη παρέχει 2 κουμπιά όσο αφορά την παρουσίαση εγγράφων το ένα είναι το κουμπί next, που σε περίπτωση που εκδοθεί εντολή λήψης και οι εγγραφές που παράλαβε ο πελάτης είναι πάνω από 14 τότε δίνεται η δυνατότητα να πατήσουμε next για να δούμε και τις υπόλοιπες και το άλλο είναι το κουμπί previous το οποίο μας παίρνει στην προηγουμένη σελίδα για να δούμε τις πιο παλιές εγγραφές. Τέλος υπάρχουν ακόμα 2 κουμπιά στην γραφική διεπαφή του πελάτη το κουμπί GET το οποίο όταν πατηθεί ο πελάτης πραγματοποιεί αίτημα λήψης από τους εξυπηρετητές, και το κουμπί APPEND που όταν πατηθεί ο πελάτης πραγματοποιεί αίτημα πρόσθεσης στους εξυπηρετητές. Για να πραγματοποιηθεί ορθά η πρόσθεση ο χρήστης πρέπει να δώσει και την εγγραφή που θέλει να προσθέσει, έτσι πάνω από το κουμπί APPEND

υπάρχει ένα text box. Όταν πατηθεί το κουμπί APPEND μέσω της εντολής AppendText.getText(); παίρνει τα περιεχόμενα του text box και τα στέλνει στους εξυπηρετητές. Στην περίπτωση αιτήματος λήψης, οι εγγραφές εμφανίζονται με την σειρά που προσαρτήθηκαν στο αντικείμενο κατάστιχου.

Πιο αναλυτικά όλα τα κουμπιά και οι ετικέτες είναι σε ένα πλαίσιο με απόλυτη διάταξη, για να μπορούμε να μετακινούμε τα widget χωρίς περιορισμούς. Μέσω την εντολής frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); επιβάλουμε στο πλαίσιο να κλείσει όταν πατηθεί το κουμπί χ πάνω δεξιά και μέσω των εντολών frame.setBounds(100,100,647,851); και (contentPane.setBackground(Color.GRAY);) ορίζουμε την τοποθεσία και το χρώμα του πλαισίου. Ακολούθως προσθέσαμε τις 17 ετικέτες η πρώτη δηλώνει τον τίτλο του πελάτη, η δεύτερη παρουσιάζει μήνυμά μόλις το αίτημα σταλεί (broadcast) στους εξυπηρετητές και η τρίτη παρουσιάζει μήνυμα μόλις λάβει τουλάχιστον f+1 ίδια reply. Τέλος οι υπόλοιπες 14 παρουσιάζουν τα περιεχόμενα του αντικειμένου κατάστιχου σε περίπτωση έκδοσης αιτήματος λήψης.

Για κάθε πελάτη ορίσαμε μια προσωρινή λίστα η οποία έχει τα περιεχόμενα του Αντικειμένου Κατάστιχου μετά από έκδοση εντολής λήψης. Τέλος υπάρχει ένας μετρητής ο οποίος μετρά πόσες εγγραφές έχει μέσα η λίστα, σε περίπτωση που υπάρχουν πάνω από 14 εγγραφές τότε ενεργοποιείται το κουμπί next το οποίο μας εμφανίζει τις επόμενες 14 εγγραφές στην λίστα αυξάνοντας και τον μετρητή. Μόλις πατηθεί μια φορά το κουμπί next ενεργοποιείται και το κουμπί previous που εμφανίζει τις προηγούμενες 15 εγγραφές μειώνοντας και τον μετρητή.



Σχήμα 6.5: Παρουσίαση γραφικής διαπροσωπείας για τον πελάτη

6.6 Περιορισμοί

Η χρήση του Window Builder είχε κάποιους περιορισμούς

- Σε περίπτωση που ήθελα να αλλάξω τα περιεχόμενα των ετικετών την ώρα της εκτέλεσης αυτό δεν μπορούσε να πραγματοποιηθεί, έτσι αναγκάστηκα να χρησιμοποιήσω ένα timer το οποίο σταματούσε προσωρινά την κανονική ροή του προγράμματος για να εκτελέσει τον κώδικα της συνάρτησης run που καλείται όταν τέλειωνε ο χρόνος του timer. Μέσω της συνάρτησης run ενημερώνονται τα περιεχόμενα τόσο του διαχειριστή όσο και των εξυπηρετητών.
- Πρέπει πρώτα να αρχικοποιηθούν όλοι οι εξυπηρετητές καθώς και οι κλάσεις SingleThreadTCPServer για να μπορέσει να τρέξει ο διαχειριστής μιας και διαβάζει από το αρχείο που δημιουργεί ο SingleThreadedTCPServer.

Κεφάλαιο 7

Παραδείγματα Εκτέλεσης

7.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα παρουσιάσουμε την ορθότητα της διπλωματικής μου εργασίας μέσα από 2 παραδείγματα. Πιο συγκεκριμένα, αφού ορίσαμε σαν Βυζαντινή συμπεριφορά εξυπηρετητή, την πρόσθεση λανθασμένης εγγραφής σε κάθε πρόσθεση θα δώσουμε ενδείξεις ως προς την ορθότητα μέσα από τα εξής παραδείγματα:

1) 5 εξυπηρετητές και 1 Βυζαντινό. Σε αυτό το παράδειγμα αναμένεται το σύστημα να εκτελεί την αναμενόμενη συμπεριφορά.
2) 5 εξυπηρετητές και 2 Βυζαντινούς. Σε αυτό το παράδειγμα αναμένεται στις λειτουργίες λήψης ο πελάτης να περιμένει επ'αόριστόν. Αυτό γίνεται επειδή δεν ικανοποιείται η συνθήκη της ατομικής πολυεκπομπής που απαιτεί $n>=3f+1$.

7.2 Παράδειγμα με 5 Εξυπηρετητές και 1 Βυζαντινό

Αρχικά θα πρέπει να τρέξουμε τις κλάσεις των εξυπηρετητών. Οι ορθοί εξυπηρετητές ξεκινούν την εκτέλεση τους μετά την εκτέλεση της ακόλουθης εντολής στο command line.

-Για πρώτο ορθό εξυπηρετητή

./runscripts/smartrun.sh bftsmart/demo/blockchain/BServer 0 192.168.10.17

-Για δεύτερο ορθό εξυπηρετητή

./runscripts/smartrun.sh bftsmart/demo/blockchain/BServer 1 192.168.10.17

-Για τρίτο ορθό εξυπηρετητή

./runscripts/smartrun.sh bftsmart/demo/blockchain/BServer 2 192.168.10.17

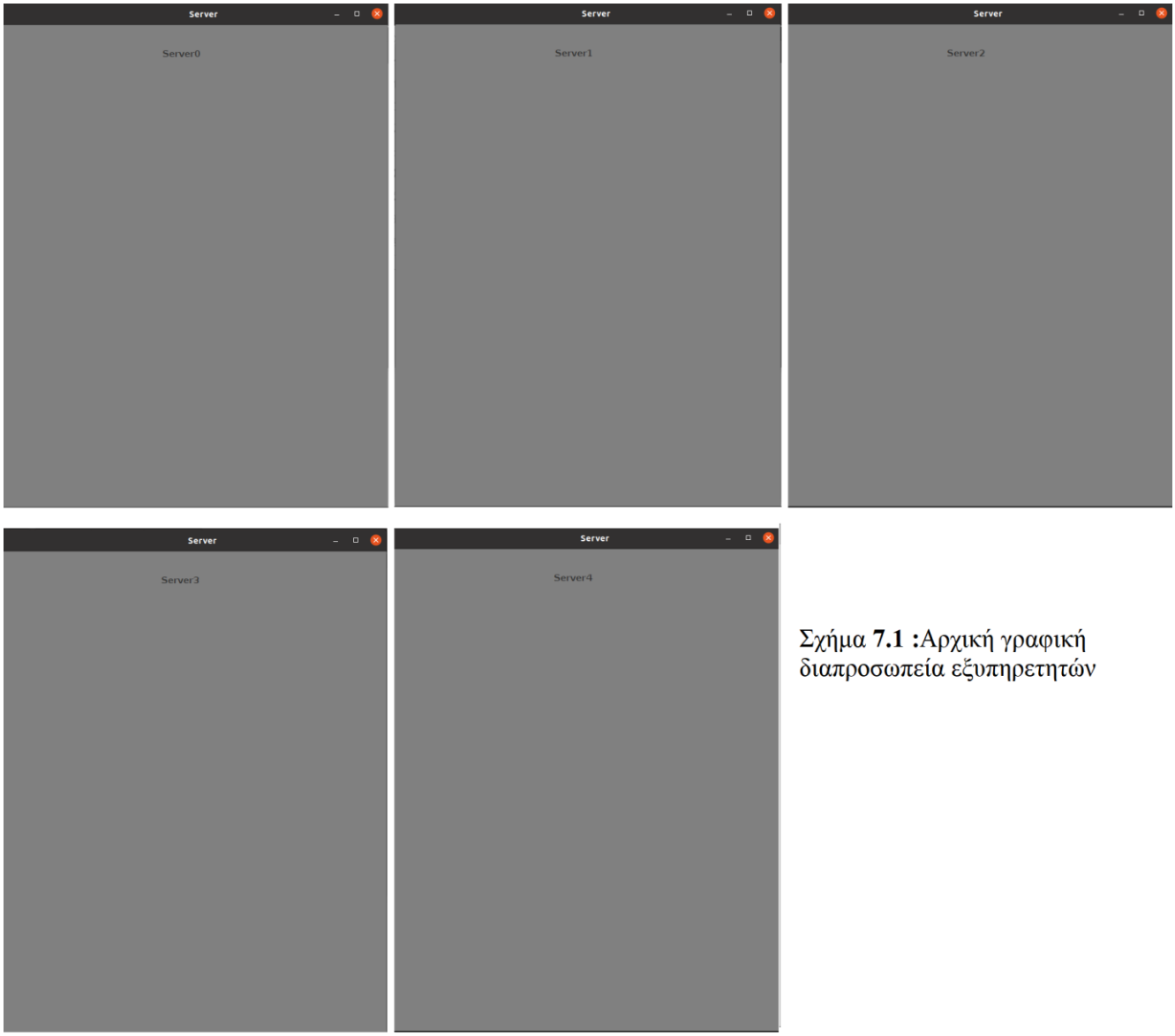
-Για τέταρτο ορθό εξυπηρετητή

./runscripts/smartrun.sh bftsmart/demo/blockchain/BServer 3 192.168.10.17

Και ο Βυζαντινός εξυπηρετητής ξεκινά την εκτέλεση του μετά την εκτέλεση της ακόλουθης εντολής

./runscripts/smartrun.sh bftsmart/demo/blockchain/BFServer 4 192.168.10.17.

Μετά την αρχικοποίηση των εξυπηρετητών εμφανίζεται η γραφική διαπροσωπεία στην οθόνη των υπολογιστών όπου εκτελέστηκαν οι πιο πάνω εντολές.



Σχήμα **7.1** :Αρχική γραφική διαπροσωπεία εξυπηρετητών

Ακολούθως τρέχουμε τις κλάσεις που εκτελούνται στον υπολογιστή του διαχειριστή, εκτελούμε τις 4 κλάσεις που είναι υπεύθυνες για την επικοινωνία με τους απομακρυσμένους εξυπηρετητές και την δημιουργία των αρχείων από τα οποία διαβάζει ο διαχειριστής για να παρουσιάσει στην οθόνη τα περιεχόμενα όλων των εξυπηρετητών.

Οι κλάσεις που είναι υπεύθυνες για την επικοινωνία με τους απομακρυσμένους εξυπηρετητές εκτελούνται με την ακόλουθη εντολή.

-Για πρώτο εξυπηρετητή

./runscripts/smartrun.sh bftsmart/demo/blockchain/SingleThreadedTCPServer 0

-Για δεύτερο εξυπηρετητή

./runscripts/smartrun.sh bftsmart/demo/blockchain/SingleThreadedTCPServer 1

-Για τρίτο εξυπηρετητή

./runscripts/smartrun.sh bftsmart/demo/blockchain/SingleThreadedTCPServer 2

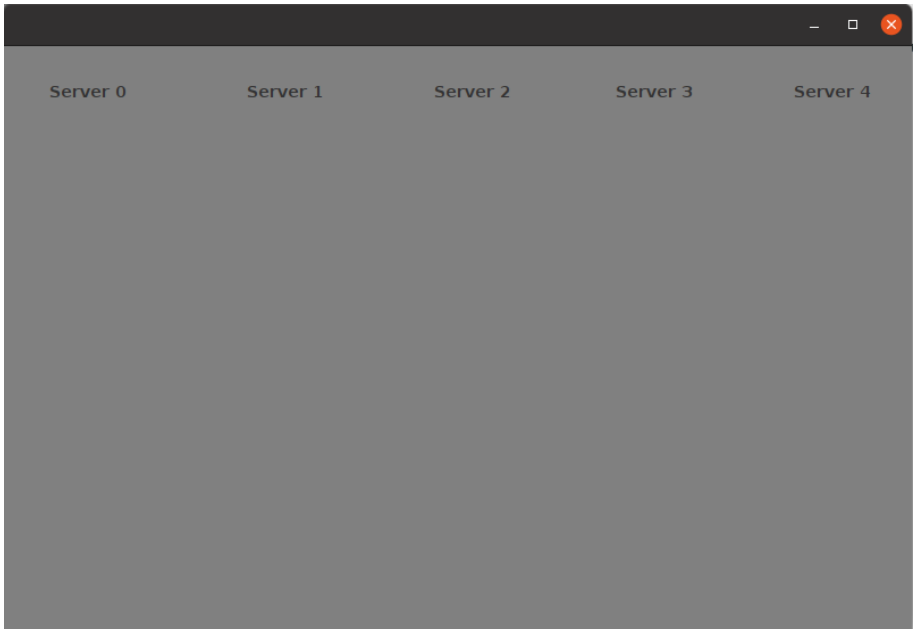
-Για τέταρτο εξυπηρετητή

./runscripts/smartrun.sh bftsmart/demo/blockchain/SingleThreadedTCPServer 3

Η κλάση για την γραφική διαπροσωπεία του διαχειριστή για σύστημα με 4 εξυπηρετητές εκτελείται με την ακόλουθη εντολή

./runscripts/smartrun.sh bftsmart/demo/blockchain/ServerAdministrator5Server 4

Και το αποτέλεσμα της πιο πάνω εντολής είναι :

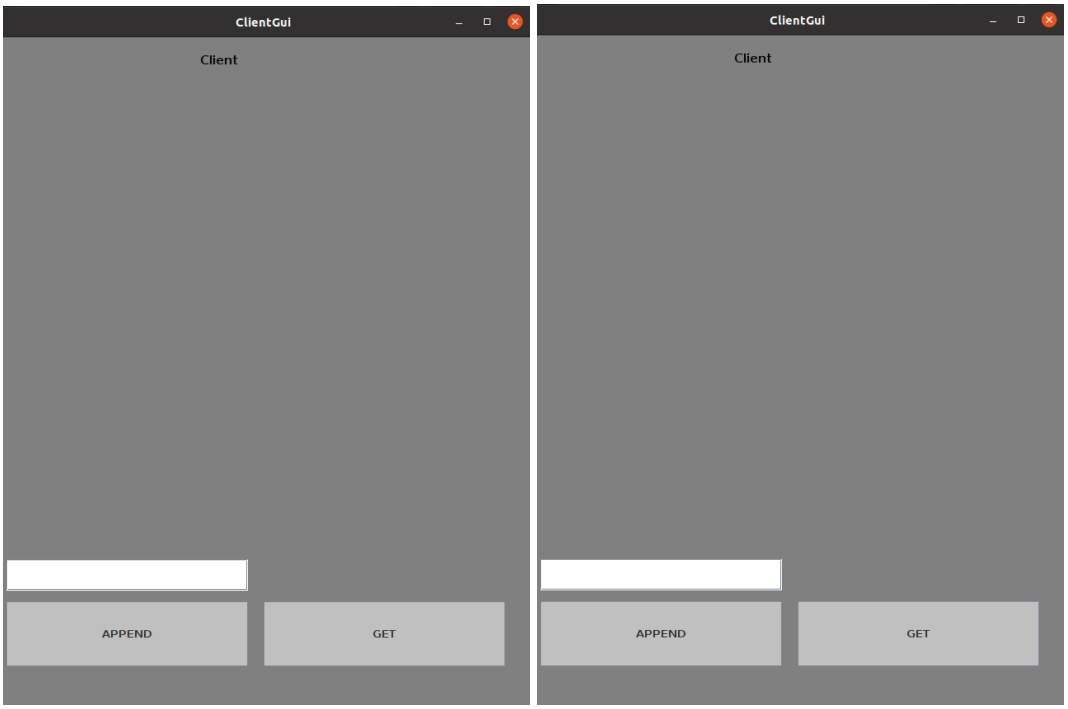


Σχήμα 7.2: Αρχική γραφική διαπροσωπεία διαχειριστή

Ακολούθως αρχικοποιούμε 2 πελάτες με τις ακόλουθες εντολές

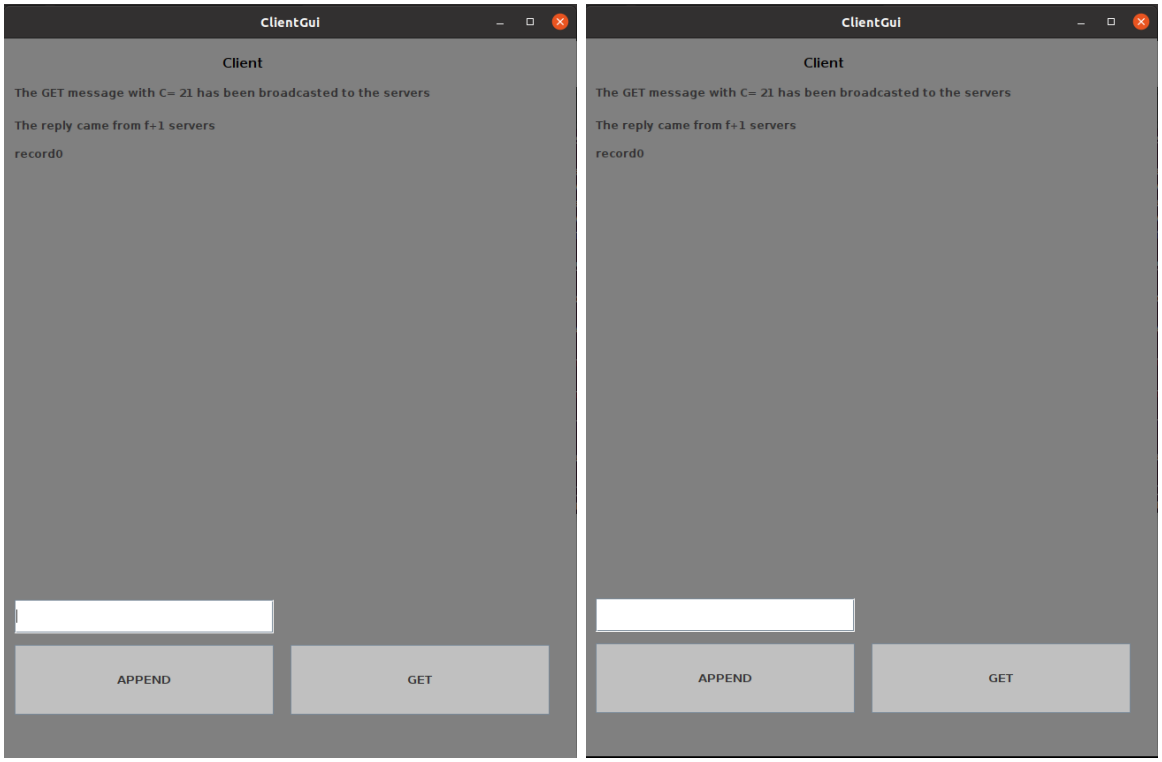
-Για πρώτο πελάτη ./runscripts/smartrun.sh bftsmart/demo/blockchain/ClientGui 1

-Για δεύτερο πελάτη ./runscripts/smartrun.sh bftsmart/demo/blockchain/ClientGui 2

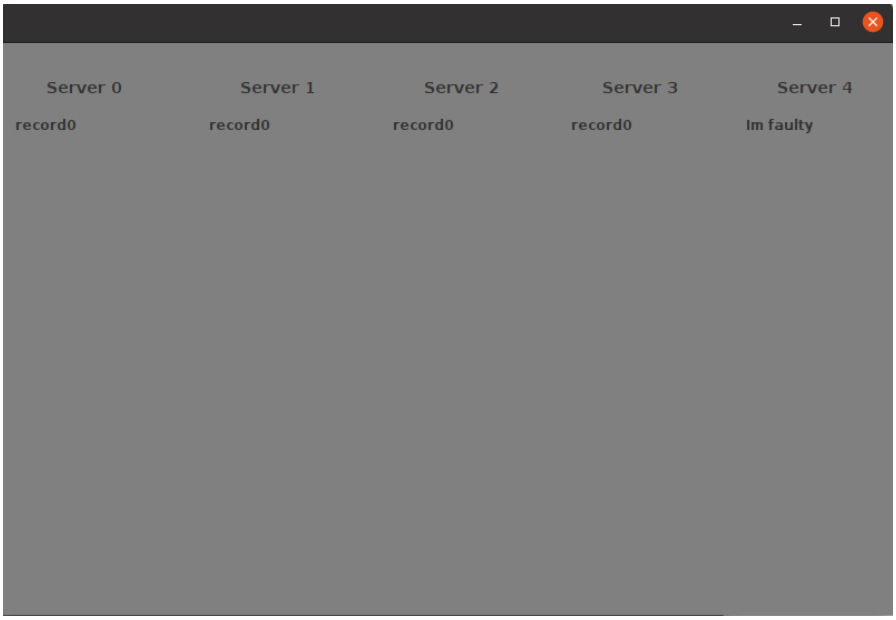


Σχήμα 7.3: Αρχική γραφική διαπροσωπεία για 2 πελάτες

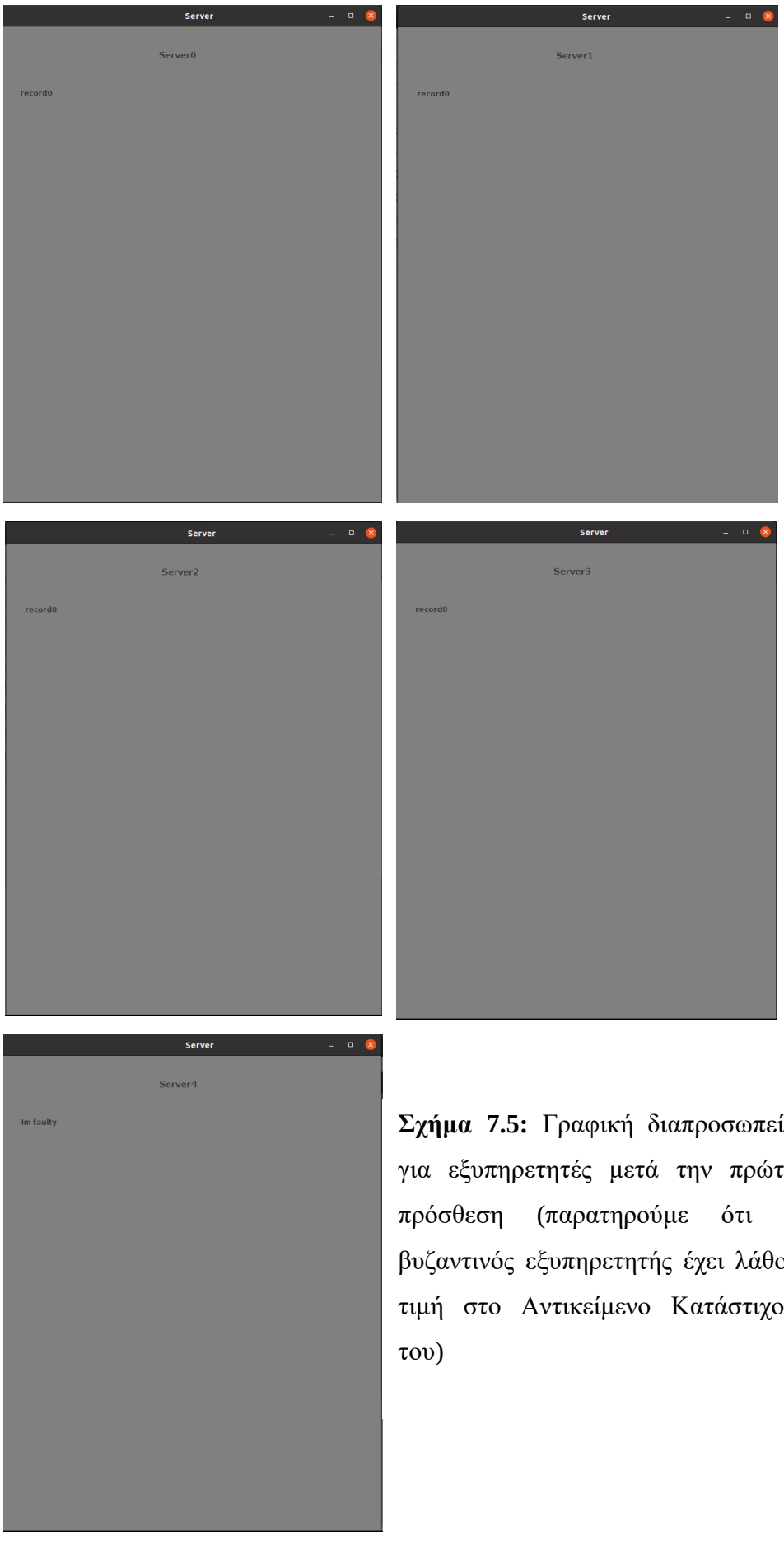
Μετά την πρώτη πρόσθεση ο καθένας θα βλέπει τα ακόλουθα αποτελέσματα. Ο πελάτης που εκτέλεσε το αίτημα, αιτήθηκε μια λειτουργία πρόσθεσης με τιμή εγγραφής την record0. Για να φανεί ότι παίρνουν τα σωστά αποτελέσματα οι πελάτες εκτελέσαμε μια εντολή λήψης (εκτέλεσε αυτή την εντολή ο κάθε πελάτης).



Σχήμα 7.4: Γραφική διαπροσωπεία για 2 πελάτες μετά την πρώτη πρόσθεση

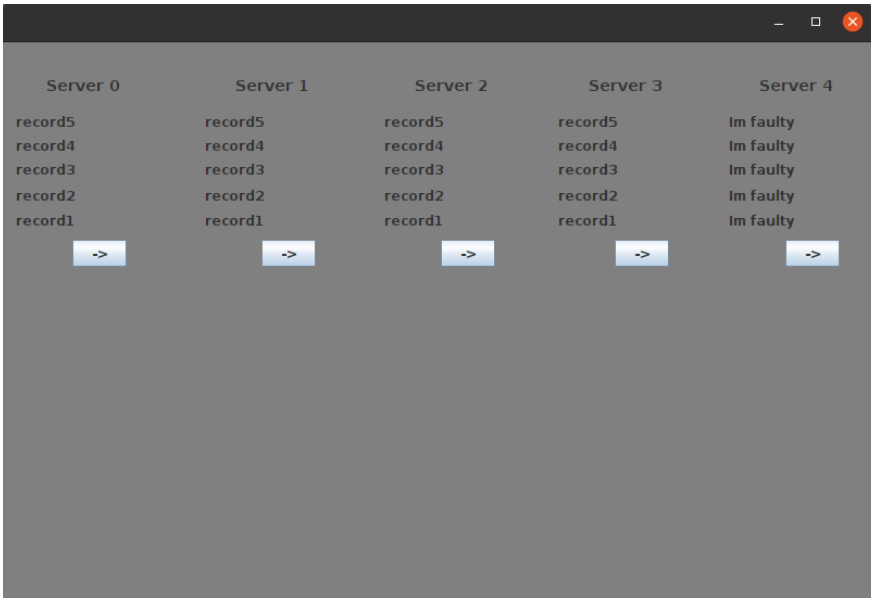


Σχήμα 7.5: Γραφική διαπροσωπεία για διαχειριστή μετά την πρώτη πρόσθεση (Παρατηρούμε ότι ο Βυζαντινός εξυπηρετητής έχει λάθος τιμή στο Αντικείμενο Κατάστιχου του)

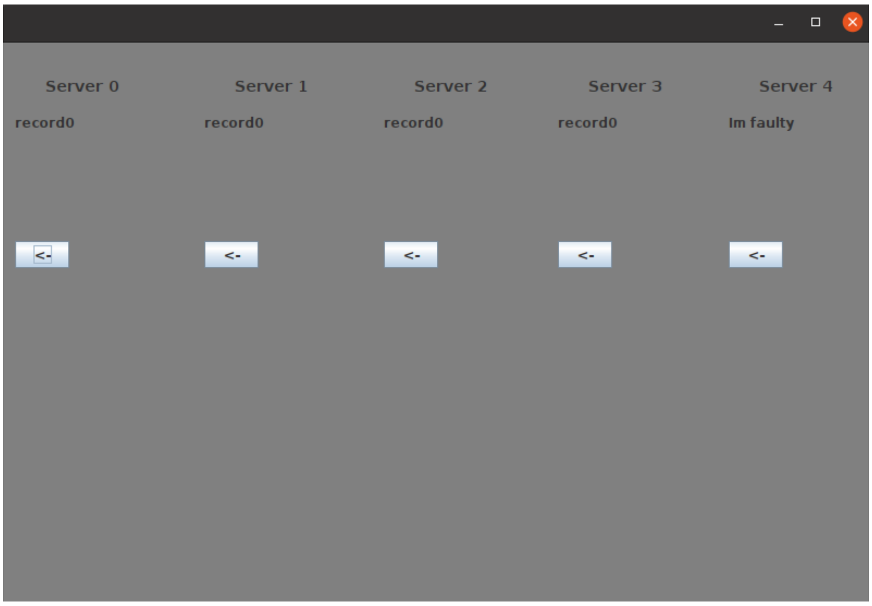


Σχήμα 7.5: Γραφική διαπροσωπεία για εξυπηρετητές μετά την πρώτη πρόσθεση (παρατηρούμε ότι ο βυζαντινός εξυπηρετητής έχει λάθος τιμή στο Αντικείμενο Κατάστιχου του)

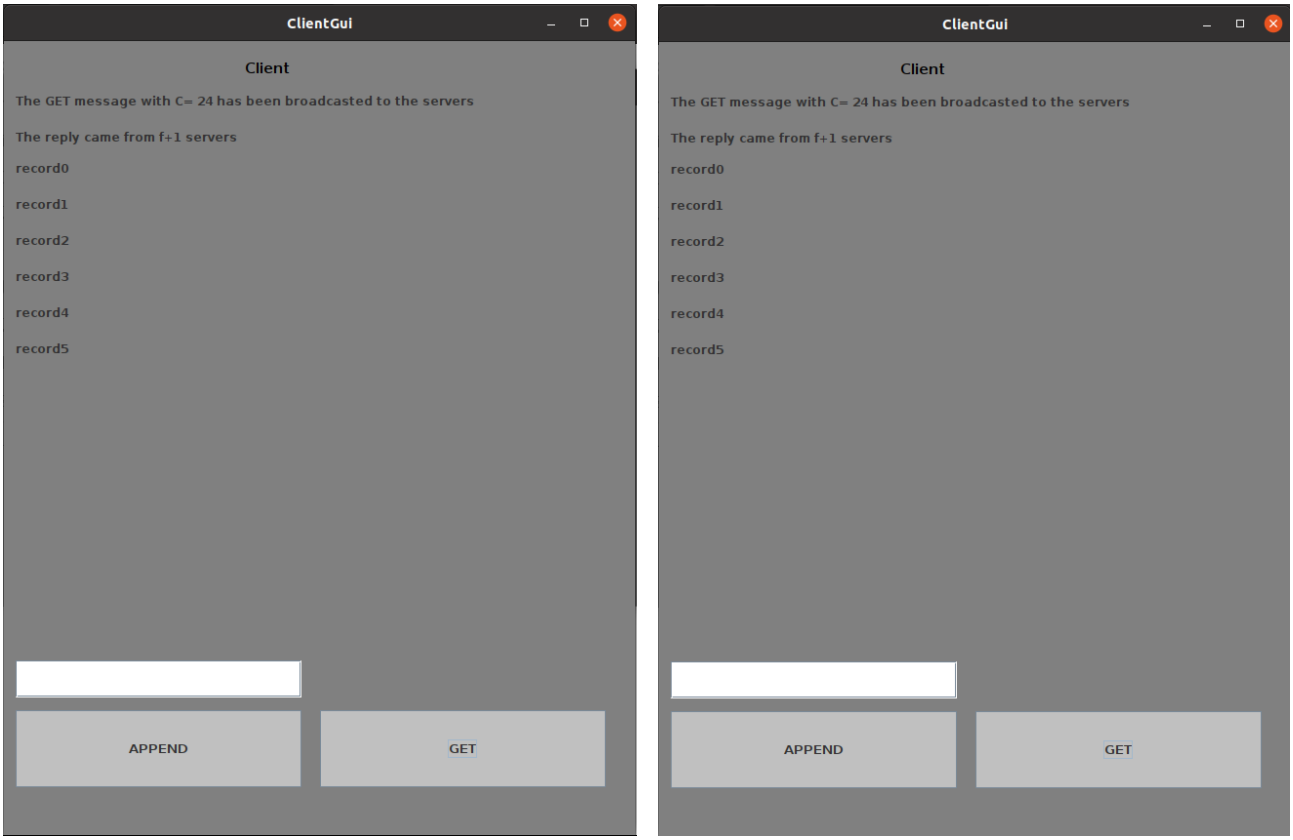
Μετά από 6 προσθέσεις



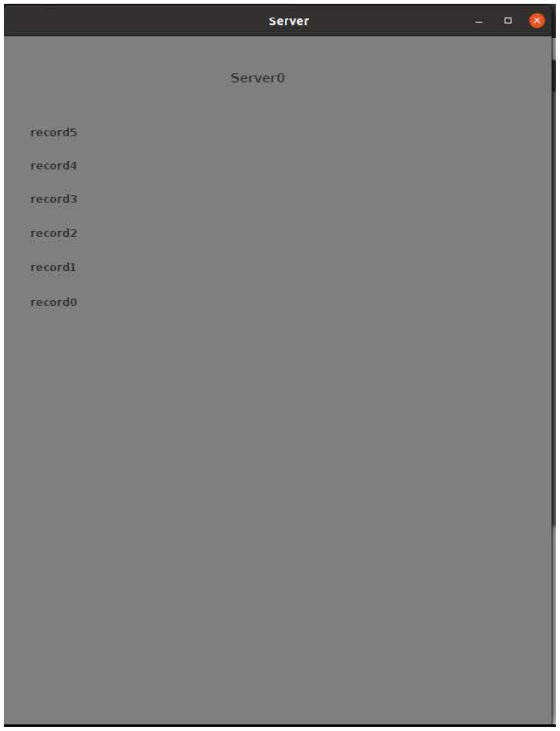
Σχήμα 7.6: Γραφική διαπροσωπεία για διαχειριστή μετά τις πρώτες 6 προσθέσεις (παρατηρούμε ότι ο Βυζαντινός εξυπηρετητής έχει λάθος τιμή στο αντικείμενο κατάστιχου του ενώ εμφανίζεται και το κουμπί για να μεταβούμε στις πιο παλιές εγγραφές).

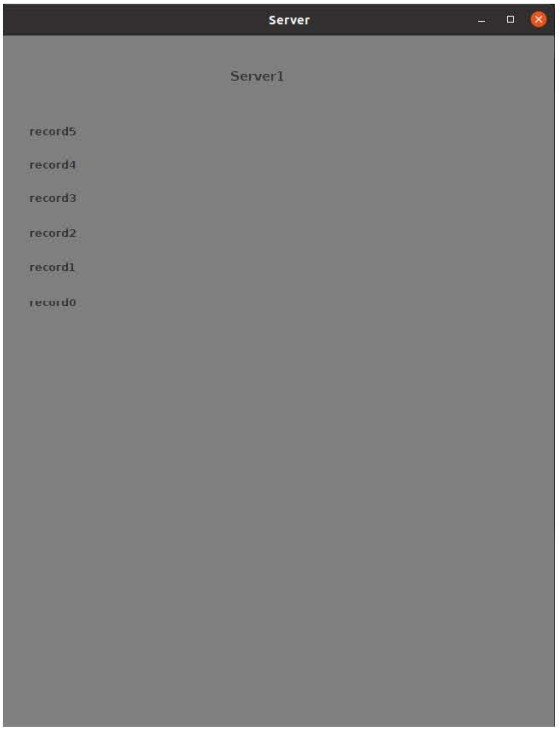


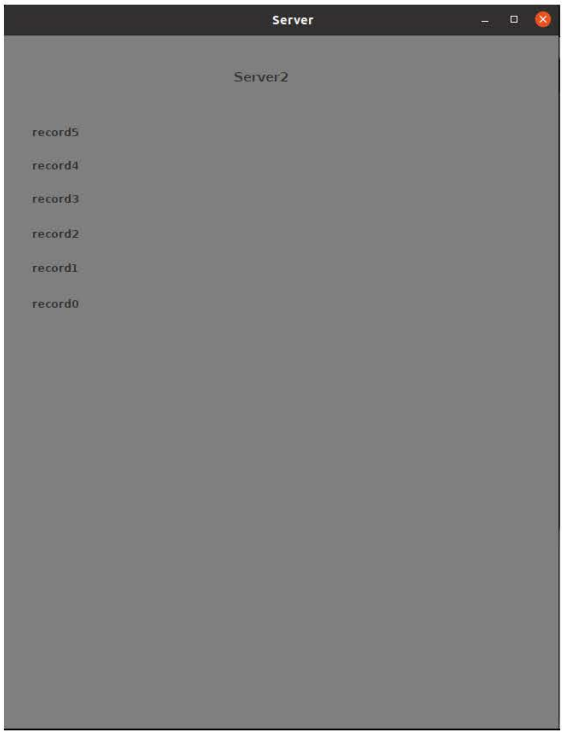
Σχήμα 7.7: Γραφική διαπροσωπεία για διαχειριστή μετά τις πρώτες 6 προσθέσεις και μετά το πάτημα του κουμπιού next

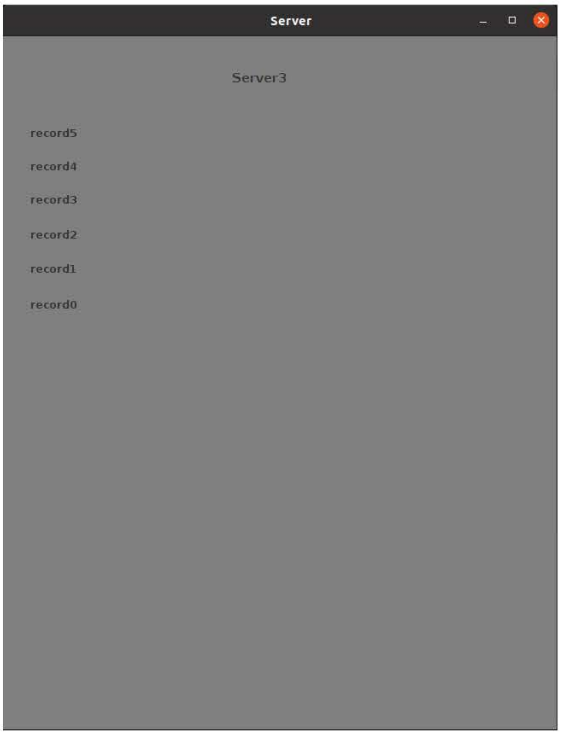


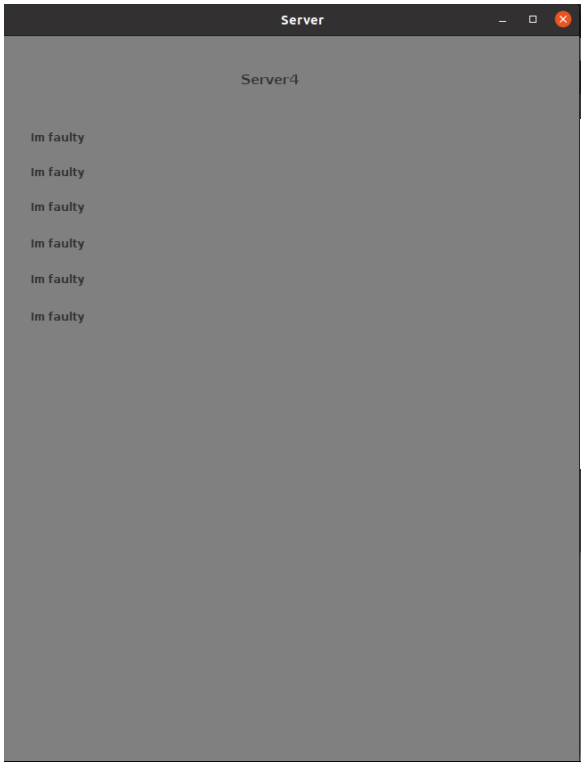
Σχήμα 7.8: Γραφική διαπροσωπεία για πελάτη μετά τις πρώτες 6 προσθέσεις και μετά την εκτέλεση αιτήματος λήψης.


Server
Server0
record5
record4
record3
record2
record1
record0


Server
Server1
record5
record4
record3
record2
record1
record0


Server
Server2
record5
record4
record3
record2
record1
record0

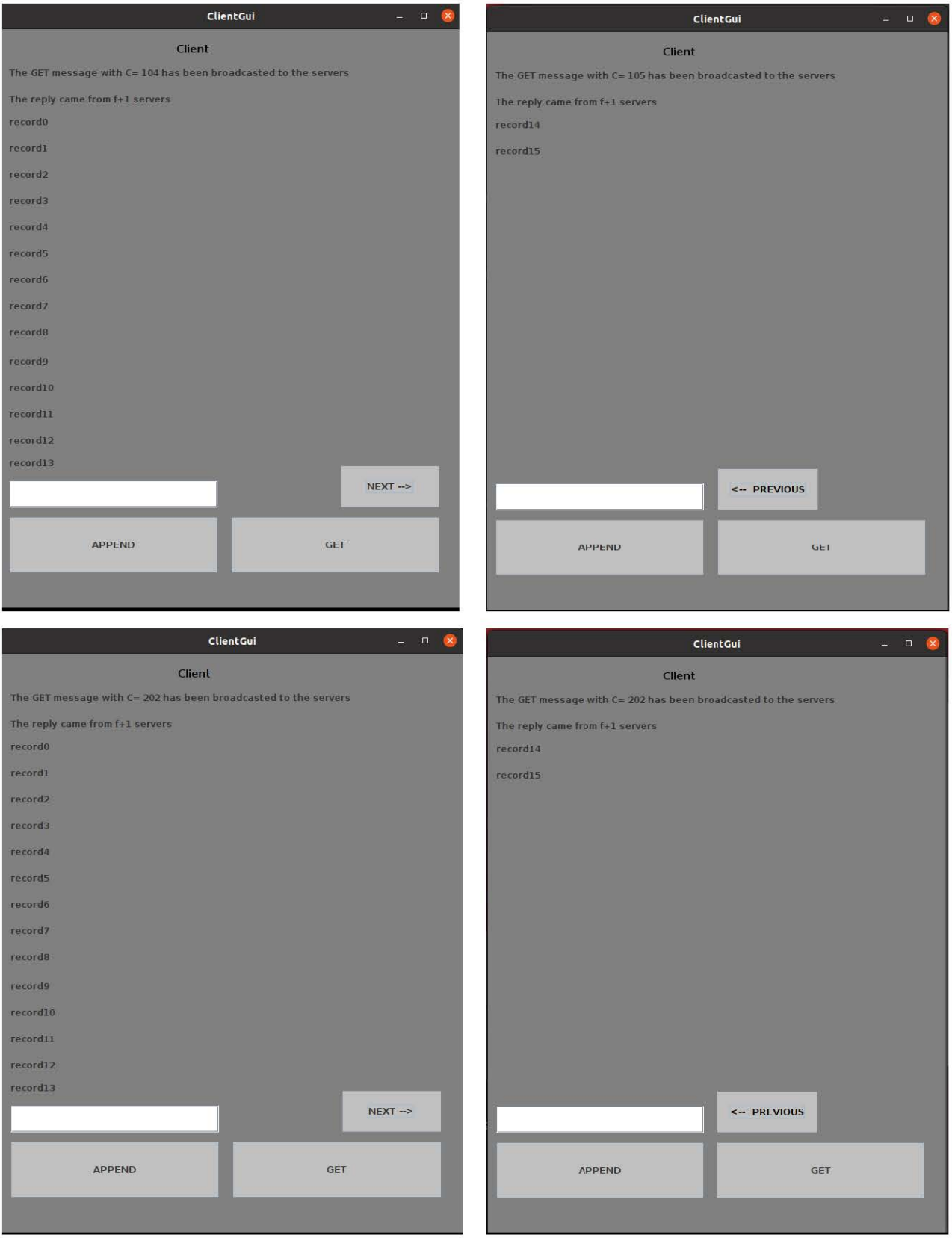

Server
Server3
record5
record4
record3
record2
record1
record0



Σχήμα 7.9: Γραφική διαπροσωπεία για εξυπηρετητές μετά τις πρώτες 6 προσθέσεις (παρατηρούμε ότι ο βυζαντινός εξυπηρετητής έχει λάθος τιμή στο Αντικείμενο Κατάστιχου του)

Μετά από 16 προσθέσεις

Σχήμα 7.10: Γραφική διαπροσωπεία για διαχειριστή μετά τις πρώτες 16 προσθέσεις



Σχήμα 7.11: Γραφική διαπροσωπεία για πελάτη μετά τις πρώτες 16 προσθέσεις


Server
Server0
record15
record14
record13
record12
record11
record10
record9
record8
record7
record6
record5
record4
record3
record2
record1


Server
Server1
record15
record14
record13
record12
record11
record10
record9
record8
record7
record6
record5
record4
record3
record2
record1


Server
Server2
record15
record14
record13
record12
record11
record10
record9
record8
record7
record6
record5
record4
record3
record2
record1

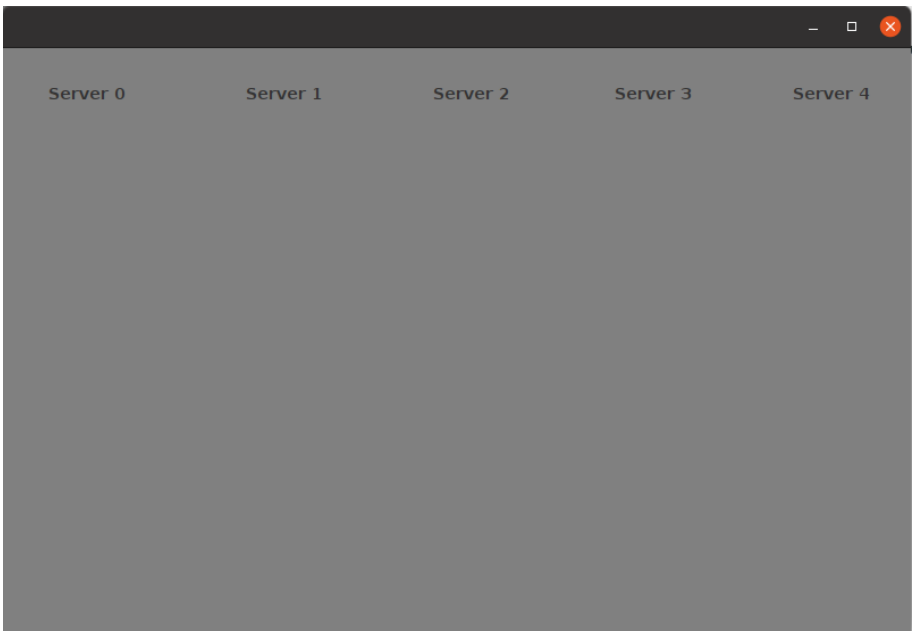

Server
Server3
record15
record14
record13
record12
record11
record10
record9
record8
record7
record6
record5
record4
record3
record2
record1

Σχήμα 7.12: Γραφική διαπροσωπεία για εξυπηρετητές μετά τις πρώτες 16 προσθέσεις (παρατηρούμε ότι ο Βυζαντινός εξυπηρετητής έχει λάθος τιμή στο Αντικείμενο Κατάστιχου του, και ότι επειδή εμφανίζονται οι 15 πιο πρόσφατες εγγραφές, το record0 δεν εμφανίζεται πλέον)

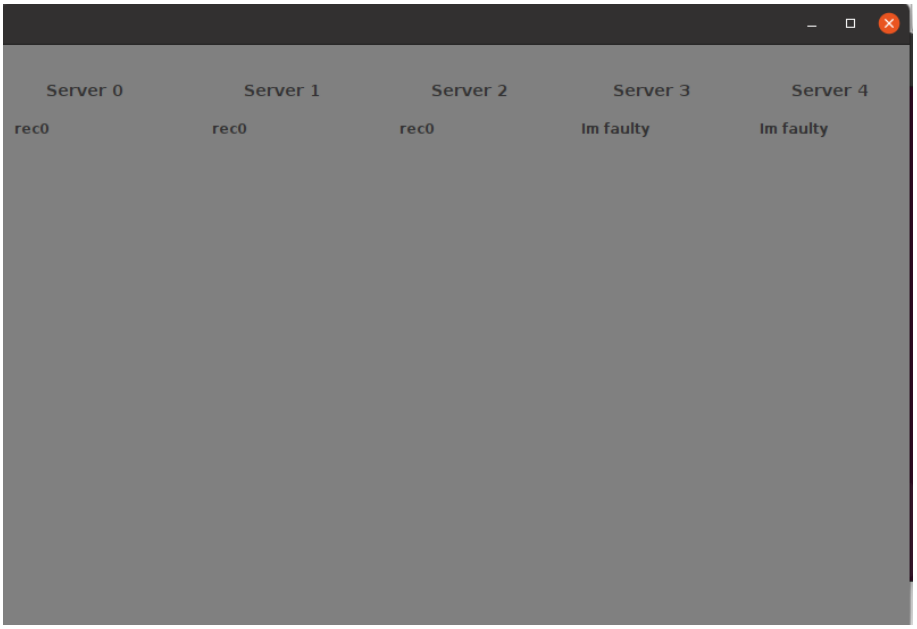
7.3 Παράδειγμα με 4 Εξυπηρετητές και 2 Βυζαντινούς

Όταν χρησιμοποιούμε 5 εξυπηρετητές και οι 2 είναι Βυζαντινοί αναμένουμε όταν ο πελάτης στείλει κάποιο αίτημα λήψης να περιμένει επ' αόριστόν. Αυτό γίνεται επειδή δεν ικανοποιείται η συνθήκη $n>=3f+1$. Στο σενάριο όπου ο πελάτης 1 στέλνει το πρώτο αίτημα για λήψη αυτό επιτυγχάνεται (τουλάχιστον $f+1$ ίδιοι εξυπηρετητές θα στείλουν την ίδια απάντηση) μιας και αρχικά όλοι οι εξυπηρετητές είναι άδειοι άρα θα στείλουν το ίδιο reply. Το ίδιο ισχύει και στο επόμενο αίτημα πού θα στείλει αίτημα πρόσθεσης, όλοι θα στείλουν ACK άρα είναι λες και ο αριθμός των Βυζαντινών εξυπηρετητών είναι $f = 0$ (τουλάχιστον $f+1$ ίδιοι εξυπηρετητές θα στείλουν την ίδια απάντηση). Στην περίπτωση όμως που το επόμενο αίτημα είναι αίτημα λήψης, τότε εκεί πραγματοποιείται η πρώτη Βυζαντινή συμπεριφορά ως προς τον πελάτη, αφού του στέλνεται αντικείμενο κατάστιχου με λάθος τιμές από 2 εξυπηρετητές, από εκεί γίνεται αντιληπτό ότι υπάρχουν 2 Βυζαντινοί εξυπηρετητές, αυτό έχει σαν αποτέλεσμα ο πελάτης να περιμένει επ' αόριστόν και να μην πραγματοποιείται σωστά η λειτουργία του συστήματος λόγω μη ικανοποίησης της συνθήκης $n>=3f+1$.

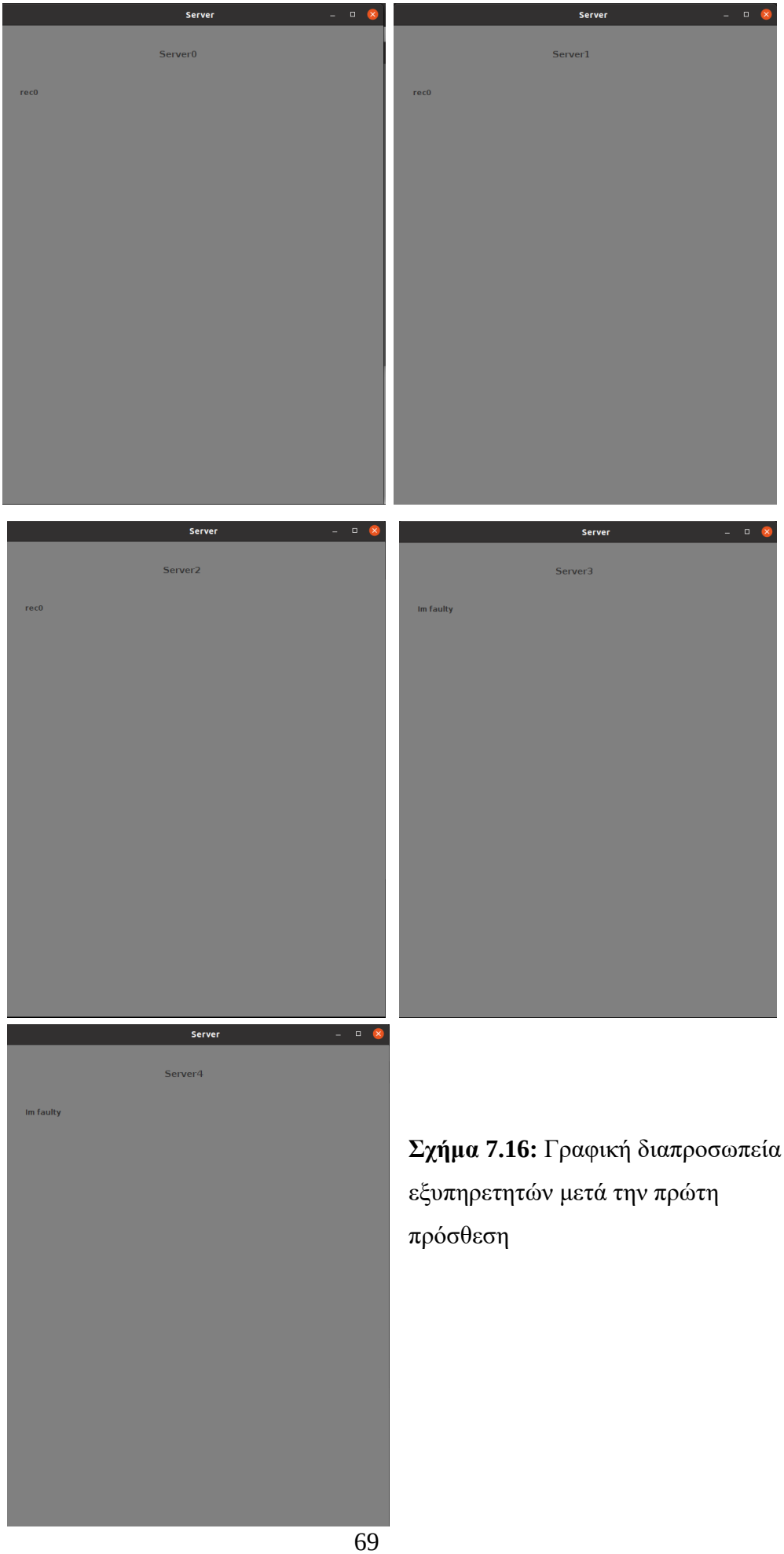
Σχήμα **7.13** :Γραφική διαπροσωπεία για εξυπηρετητές αρχικά



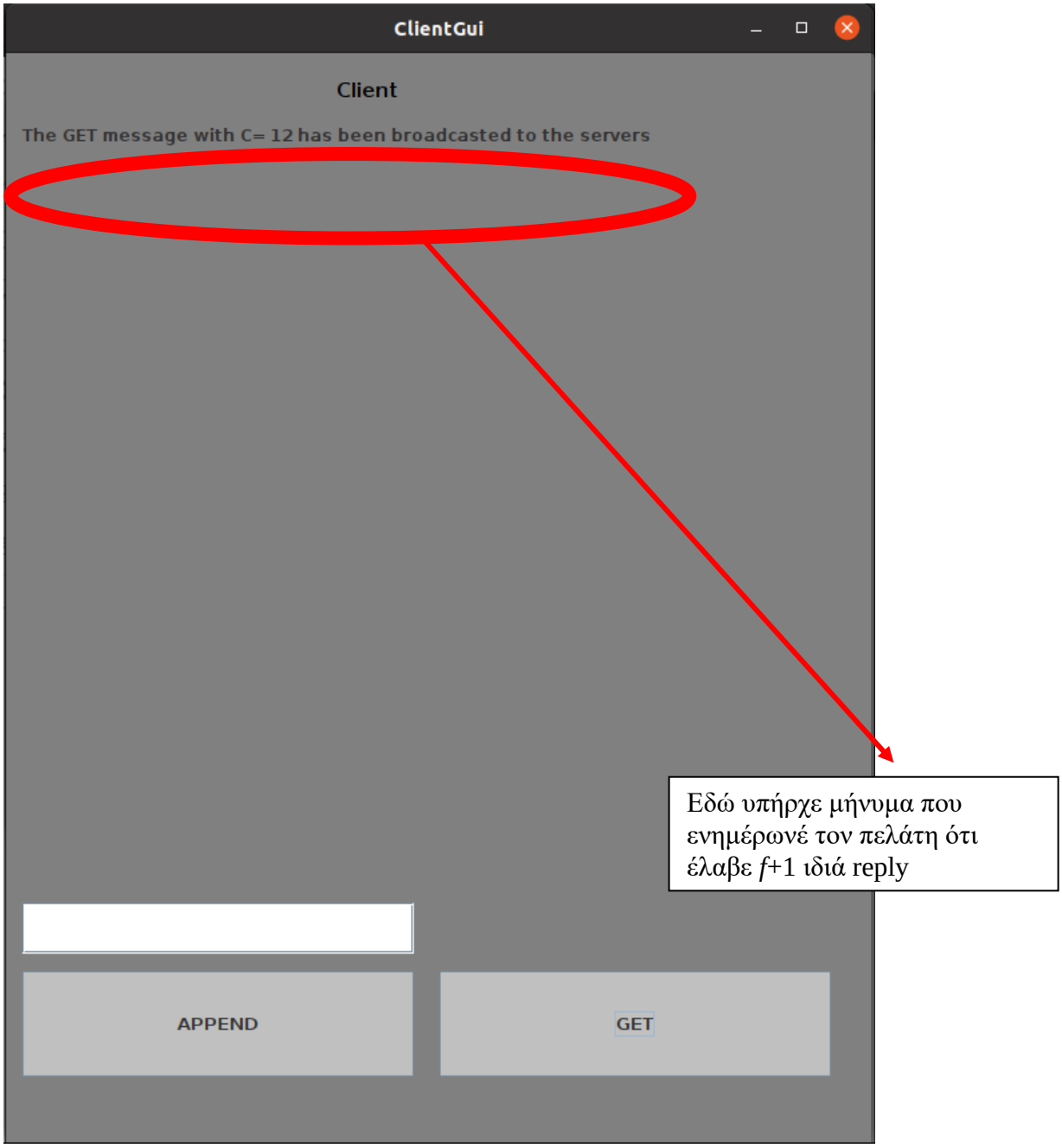
Σχήμα 7.14: Γραφική διαπροσωπεία για διαχειριστή



Σχήμα 7.15: Γραφική διαπροσωπεία για διαχειριστή μετά την πρώτη πρόσθεση



Σχήμα 7.16: Γραφική διαπροσωπεία εξυπηρετητών μετά την πρώτη πρόσθεση



Σχήμα 7.17: Γραφική διαπροσωπεία για πελάτη. Μετά το αίτημα λήψης βλέπουμε ότι ο πελάτης στα μηνύματα λαμβάνει μόνο ότι το μήνυμα έγινε broadcast και δεν λαμβάνει το μήνυμα ότι λήφθηκαν τουλάχιστον $f+1$ ίδιες απαντήσεις από διαφορετικούς εξυπηρετητές, άρα ούτε και το σύνολο των νέων εγγράφων λαμβάνεται. Αυτό γίνεται επειδή δεν ικανοποιείται η συνθήκη $n>=3f+1$.

7.4 Έλεγχος Ορθότητας Υλοποίησης

Για τον έλεγχο ορθότητας του συστήματος τρέξαμε κάποια Unit test. Το σύστημα πέρασε με επιτυχία όλα τα test, ενώ λόγο της γραφικής διαπροσωπείας ο έλεγχος ορθότητας ήταν πιο εύκολος. Μέσω της γραφικής διαπροσωπείας είχαμε οπτική επαφή με τα περιεχόμενα κάθε εξυπηρετητή και δεν χρειαζόταν να ανατρέχουμε σε log files. Τα σενάρια τα που ελέγχθηκαν ήταν τα ακόλουθα:

Αριθμός Σεναρίου	Συνολικός Αριθμός Εξυπηρετητών	Συνολικός Αριθμός Βυζαντινών Εξυπηρετητών	Αποτέλεσμα	Αναμενόμενο Αποτέλεσμα
1	4	1		
2	4	2		
3	4	3		
4	4	4		
5	5	1		
6	5	2		
7	5	3		
8	5	4		
9	6	1		
10	6	2		
11	6	3		
12	6	4		
13	7	1		
14	7	2		
15	7	4		
16	7	6		
17	8	1		
18	8	2		
19	8	3		
20	8	6		

21	9	2		
22	9	3		
23	9	4		
24	9	7		
25	10	1		
26	10	2		
27	10	3		
28	10	4		

*Με κόκκινο χρώμα είναι τα σενάρια τα οποία το σύστημα δεν λειτουργεί (επειδή δεν ικανοποιείται η συνθήκη $n>=3f+1$)

*Με πράσινο χρώμα είναι τα σενάρια τα οποία το σύστημα λειτουργεί

Κεφάλαιο 8

Συμπεράσματα

8.1 Περίληψη

Εν συντομία, στην παρούσα διπλωματική εργασία μελετήθηκε και υλοποιήθηκε ένα κατανεμημένο σύστημα Κατάστιχων ανθεκτικό σε Βυζαντινά σφάλματα, το οποίο είναι κτισμένο πάνω σε υπηρεσία ατομικής πολυεκπομπής ανθεκτική σε Βυζαντινά σφάλματα. Μέσω της υπηρεσίας καταφέραμε οι εγγραφές να αποθηκεύονται με την σωστή σειρά στα Αντικείμενα Κατάστιχου, τα αιτήματα να εξυπηρετούνται από όλους τους εξυπηρετητές ως προς την σειρά που παραλήφθηκαν και να στέλνονται οι απαντήσεις με την ανάλογη σειρά στους πελάτες. Επίσης, για το σύστημα κληθήκαμε να υλοποιήσουμε και την γραφική διαπροσωπεία, έτσι ώστε να είναι εύχρηστο από τους απλούς χρήστες μιας και η χρήση της γραμμής εντολής απαιτεί προηγμένη γνώση. Τέλος, για την υπηρεσία ατομικής πολυεκπομπης χρειάστηκε η μελέτη και χρήση του εργαλείου BFT-Smart το οποίο μας παρείχε όλες τις εγγυήσεις της υπηρεσίας ατομικής πολυεκπομπής αλλά και η μελέτη των υποδοχών (Sockets). Μέσω των υποδοχών (Sockets) πετύχαμε το σύστημα μας να δουλεύει σε ένα πλήρες κατανεμημένο σύστημα όπου οι πελάτες, οι εξυπηρετητές και ο διαχειριστής μπορούν να βρίσκονται σε διαφορετικές μηχανές.

Μέσα από την εργασία είχα την ευκαιρία να παρατηρήσω την σημαντικότητα ενός κατανεμημένου συστήματος και να καταλάβω πως τέτοιοι αλγόριθμοι είναι υλοποιήσιμοι σε ένα ασύγχρονο περιβάλλον που υπάρχει πιθανότητα να προκύψουν σφάλματα.

8.2 Δυσκολίες και Προκλήσεις

Καθ' όλη την διάρκεια της εργασίας υπήρξαν αρκετές δυσκολίες. Αρχικά το BFT-Smart απαιτούσε χρήση αλγορίθμων οι οποίοι από προεπιλογή ήταν αποκλεισμένοι από την java, έτσι έπρεπε να μπούμε στο αρχείο java.security της έκδοσης Java που χρησιμοποιούμε και να βάλουμε σε σχόλιο τις γραμμές με τους απενεργοποιημένους αλγορίθμους. Για παράδειγμα αν χρησιμοποιούμε την έκδοση Java 1.8 πάμε στο αρχείο
C:\Program Files\Java\jdk1.8.0_181\jre\lib\security\java.security και βάζουμε σε σχόλια τα «jdk.tls.disabledAlgorithms=SSLv3, TLSv1, TLSv1.1, RC4, DES, MD5withRSA, DH keySize lessThan 1024, EC keySize lessThan 224, 3DES_EDE_CBC, anon, NULL, include jdk.disabled.namedCurves» (υπάρχει link με το πως γίνεται αυτό [8]).

Επίσης κατά τους ελέγχους που πραγματοποιήθηκαν σε Ubuntu παρατηρήθηκε το εξής σφάλμα:

```
Exception in thread "main" java.awt.AWTError: Assistive Technology not found:
com.sun.java.accessibility.AccessBridge
at java.awt.Toolkit.loadAssistiveTechnologies(Toolkit.java:775)
at java.awt.Toolkit.getDefaultToolkit(Toolkit.java:861)
at java.awt.Window.getToolkit(Window.java:1127)
at java.awt.Window.init(Window.java:369)
at java.awt.Window.(Window.java:407)
at java.awt.Frame.(Frame.java:402)
at java.awt.Frame.(Frame.java:367)
at javax.swing.JFrame.(JFrame.java:163)
at FirstJavaProject.(FirstJavaProject.java:7)
at FirstJavaProject.main(FirstJavaProject.java:5)
```

Αυτό το σφάλμα εμφανίζεται όταν έχουμε εγκατεστημένη μια headless JRE έκδοση. Το headless JRE είναι υποσύνολο του πλήρους JRE χωρίς χαρακτηριστικά γραφικής διαπροσωπείας όπως η υποστήριξη σε βοηθητικές τεχνολογίες (assistive technologies). Για να διορθωθεί το πρόβλημα εκτελούμε την εντολή
sudo vim /etc/java-8-openjdk/accessibility.properties
και βάζουμε σε σχόλιο την γραμμή
#assistive_technologies=org.GNOME.Accessibility.AtkWrapper [11]

Ένα άλλο πρόβλημα που δεν συναντήθηκε σε εμένα αλλά πιθανόν να εμφανισθεί στους χρήστες, είναι η απαίτηση του BFT-Smart να υπάρχει Java-Runtime-Environment (JRE) 1.8 ή πιο πρόσφατο.

Όσον αφορά τα προβλήματα υλοποίησης (αν και δεν θεωρείται ακριβώς πρόβλημα αλλά είναι καλό να αναφερθεί), στην βιβλιοθήκη BFT-Smart, όταν δηλωθεί αριθμός Βυζαντινών εξυπηρετητών μικρότερος κατά 2,1 η 0 ($n<=f+2$) από τον συνολικό αριθμό εξυπηρετητών τότε η επικοινωνία μεταξύ όλων των εξυπηρετητών δεν αρχικοποιείται ποτέ.

Τέλος, αν και στο εγχειρίδιο [5] του BFT-Smart υπήρχαν σαφείς οδηγίες μου πήρε αρκετό διάστημα να το κατανοήσω και να μάθω να το χρησιμοποιώ. Επίσης πέρα από την αρκετή μελέτη της βιβλιοθήκης και των άρθρων έπρεπε να εξοικειωθώ με το εργαλείο WindowBuilder μέσω του οποίου έκανα την γραφική διαπροσωπεία αλλά και να μελετήσω ποιες δομές θα μου παρείχαν καλύτερα αποτελέσματα και θα αντιπροσώπευαν καλυτέρα το Αντικείμενο Κατάστιχου κάθε εξυπηρετητή.

8.3 Μελλοντική Εργασία

Η διπλωματική θα μπορούσε να επεκταθεί ακόμα περισσότερο. Αρχικά θα μπορούσαμε στην υλοποίηση να προσθέσουμε και τους Βυζαντινούς πελάτες, πιο συγκεκριμένα θα κάναμε την υπόθεση ότι γνωρίζουμε εκ των προτέρων τον αριθμό των Βυζαντινών πελατών, έστω t έτσι για να πραγματοποιηθεί κάποιο αίτημα θα έπρεπε κάθε εξυπηρετητής να λάβει $t+1$ ίδια αιτήματα από διαφορετικούς πελάτες.

Επίσης θα μπορούσαμε να επεκτείνουμε τον αλγόριθμο για να πραγματοποιεί ατομικές προσθέσεις (atomic append) [2] αυτό θα μπορούσε να γίνει με 2 αντικείμενα κατάστιχου όπου μια πρόσθεση θα γινόταν στο Αντικείμενο Κατάστιχου 1 μόνο αν έφθανε και η αντίστοιχη πρόσθεση και για το Αντικείμενο Κατάστιχου 2.

Ακόμη θα μπορούσαν τα 2 αρχεία διαμόρφωσής να γεμίζουν αυτόματα με τις τιμές των IP διευθύνσεων και να μην χρειάζεται ο χρήστης να τα γράφει.

Η υλοποίηση της ΑΔΕ δουλεύει και έχει ελεγχθεί για τοπικά δίκτυα (LAN). Ακολουθώντας την εργασία στο [6], η υλοποίηση της ΑΔΕ μπορεί να επεκταθεί και να μετατραπεί έτσι ώστε να δουλεύει αποδοτικά και στο διαδίκτυο (σε δίκτυο WAN).

Βιβλιογραφία

[1] Antonio Fernandez Anta, Vicent Cholvi, Chryssis Georgiou, Nicolas Nicolaou, Michel Raynal "Atomic Appends in Asynchronous Byzantine Distributed Ledgers "

[2] Antonio Fernandez Anta, Jennifer L Welch, Kishori Konwar, Chryssis Georgiou, Nicolas Nicolaou, Michel Raynal "Distributed Computing Column 70 Formalizing and Implementing Distributed Ledger Objects".

[3] Antonio Fernandez Anta, Vicent Cholvi,Chryssis Georgiou, Nicolas Nicolaou, Michel Raynal "Appending Atomically in Byzantine Distributed Ledgers"

[4] Alysson Bessani, João Sousa Eduardo E. P. Alchieri "State Machine Replication for the Masses with BFT-SMART "

[5] Alysson Bessani, João Sousa Eduardo E. P. Alchieri https://github.com/bft-smart/library/wiki/Getting-Started-with-BFT-SMaRt

[6] Alysson Bessani, João Sousa "Separating the WHEAT from the Chaff: AnEmpirical Design for Geo-Replicated State Machines"

[7] BFT-Smart Available: https://github.com/bft-smart/library

[8] CodeSimple https://www.youtube.com/watch?v=xSejtYOh4C0&ab_channel=CodeSimple

[9] Window Builder Available: https://www.eclipse.org/windowbuilder/#:~:text=WindowBuilder%20is%20composed%20of%20SWT,will%20be%20generated%20for%20you

[10] BFT-Smart Configuration Parameters Available: https://github.com/bft-smart/library/wiki/BFT-SMaRt-Configuration

[11] AskUbuntu: https://askubuntu.com/questions/695560/assistive-technology-not-found-awterror

[12] StackPath: https://blog.stackpath.com/distributed-system/

Παράρτημα Α

```
1 package bftsmart.demo.blockchain;
2 import java.io.File;
3 import java.io.FileWriter;
4 import java.io.ByteArrayInputStream;
5 import java.io.ByteArrayOutputStream;
6 import java.io.IOException;
7 import java.io.ObjectInput;
8 import java.io.ObjectInputStream;
9 import java.io.ObjectOutput;
10 import java.io.ObjectOutputStream;
11 import java.util.Map;
12 import java.util.Set;
13 import java.util.ArrayList;
14 import java.util.List;
15 import java.util.TreeMap;
16 import java.util.TreeSet;
17 import java.util.logging.Level;
18 import java.util.logging.Logger;
19 import javax.swing.JFrame;
20 import javax.swing.JPanel;
21 import javax.swing.border.EmptyBorder;
22 import java.awt.Color;
23 import javax.swing.JLabel;
24 import javax.swing.SwingConstants;
25 import java.util.ArrayList;
26 import java.util.List;
27 import java.awt.Font;
28 import java.util.Timer;
29 import java.util.TimerTask;
30 import java.io.BufferedReader;
31 import java.io.DataOutputStream;
32 import java.io.InputStreamReader;
33 import java.net.Socket;
34 import java.util.Date;
35 import java.util.Scanner;
36
37
38 import javax.swing.JButton;
39 import java.awt.event.ActionListener;
40 import java.util.List;
41 import java.awt.event.ActionEvent;
42
43 import bftsmart.tom.MessageContext;
44 import bftsmart.tom.ServiceReplica;
45 import bftsmart.tom.server.defaultservices.DefaultSingleRecoverable;
46
47 public class BServer<K, V> extends DefaultSingleRecoverable {
48
49
50         private Logger logger;
51         private String FileName;
52         private int    PortId;
53         private String IpAddAdmin;
54         private File myObj;
55         private List<String> LedgerFinal;
56         private JPanel contentPane;
57         private JLabel Label1;
58         private JLabel Label2;
59         private JLabel Label3;
60         private JLabel Label4;
61         private JLabel Label5;
62         private JLabel Label6;
63         private JLabel Label7;
64         private JLabel Label8;
65         private JLabel Label9;
66         private JLabel Label10;
67         private JLabel Label11;
68         private JLabel Label12;
69         private JLabel Label13;
70         private JLabel Label14;
71         private JLabel Label15;
72
73         public BServer(int id,String IpAddressAdmin,JFrame frame) {
74
75                 FileName="LogFile"+id+"Server.txt";
76                 PortId=id;
77                 IpAddAdmin= IpAddressAdmin;
78                 try {
79                         myObj = new File(FileName);
80                         if (myObj.createNewFile()) {
81                                 System.out.println("File created: " + myObj.getName());
82                         } else {
83                                 System.out.println("File already exists.");
84                         }
85                 }
86                 catch (IOException e) {
```

```
87                      System.out.println("An error occurred.");
88                      e.printStackTrace();
89              }
90
91              System.out.println("The ip of the administrator is " + IpAddAdmin);
92              LedgerFinal = new ArrayList<String>();
93              logger = Logger.getLogger(BServer.class.getName());
94              new ServiceReplica(id, this, this);
95
96              frame.setBackground(Color.GRAY);
97              frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
98              frame.setBounds(100, 100, 649, 850);
99              contentPane = new JPanel();
100             contentPane.setBackground(Color.GRAY);
101             contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
102             frame.setContentPane(contentPane);
103             contentPane.setLayout(null);
104
105             JLabel lblNewLabel = new JLabel("Server # ");
106             lblNewLabel.setText("Server"+id);
107             lblNewLabel.setFont(new Font("Tahoma", Font.BOLD, 15));
108             lblNewLabel.setBounds(269, 27, 112, 44);
109             contentPane.add(lblNewLabel);
110
111             JLabel Label1 = new JLabel("");
112             Label1.setBounds(33, 105, 200, 16);
113             contentPane.add(Label1);
114
115             JLabel Label2 = new JLabel("");
116             Label2.setBounds(33, 144, 200, 16);
117             contentPane.add(Label2);
118
119             JLabel Label3 = new JLabel("");
120             Label3.setBounds(33, 183, 200, 16);
121             contentPane.add(Label3);
122
123             JLabel Label4 = new JLabel("");
124             Label4.setBounds(33, 224, 200, 16);
125             contentPane.add(Label4);
126
127             JLabel Label5 = new JLabel("");
128             Label5.setBounds(33, 264, 200, 16);
129             contentPane.add(Label5);
130
131             JLabel Label6 = new JLabel("");
132             Label6.setBounds(33, 306, 200, 16);
133             contentPane.add(Label6);
134
135             JLabel Label7 = new JLabel("");
136             Label7.setBounds(33, 345, 200, 16);
137             contentPane.add(Label7);
138
139             JLabel Label8 = new JLabel("");
140             Label8.setBounds(33, 388, 200, 16);
141             contentPane.add(Label8);
142
143             JLabel Label9 = new JLabel("");
144             Label9.setBounds(33, 428, 200, 16);
145             contentPane.add(Label9);
146
147             JLabel Label10 = new JLabel("");
148             Label10.setBounds(33, 468, 200, 16);
149             contentPane.add(Label10);
150
151             JLabel Label11 = new JLabel("");
152             Label11.setBounds(33, 507, 200, 16);
153             contentPane.add(Label11);
154
155             JLabel Label12 = new JLabel("");
156             Label12.setBounds(33, 548, 200, 16);
157             contentPane.add(Label12);
158
159             JLabel Label13 = new JLabel("");
160             Label13.setBounds(33, 587, 200, 16);
161             contentPane.add(Label13);
162
163             JLabel Label14 = new JLabel("");
164             Label14.setBounds(33, 629, 200, 16);
165             contentPane.add(Label14);
166
167             JLabel Label15 = new JLabel("");
168             Label15.setBounds(33, 670, 200, 16);
169             contentPane.add(Label15);
170
171             Timer timer = new Timer();
```

```
172             TimerTask task = new TimerTask() {
173                     int counterian=0;
174                     @Override
175                     public void run() {
176
177                                     if(LedgerFinal.size()>=1){
178                                     if(LedgerFinal.get(LedgerFinal.size()-1)!=null);{
179                                             Label1.setText(LedgerFinal.get(LedgerFinal.size()-1));
180
181                                     }
182                                     }
183
184                                     if(LedgerFinal.size()>=2){
185                                     if(LedgerFinal.get(LedgerFinal.size()-2)!=null);{
186                                             Label2.setText(LedgerFinal.get(LedgerFinal.size()-2));
187                                     }
188                                     }
189
190                                     if(LedgerFinal.size()>=3){
191                                     if(LedgerFinal.get(LedgerFinal.size()-3)!=null);{
192                                             Label3.setText(LedgerFinal.get(LedgerFinal.size()-3));
193                                     }
194                                     }
195
196                                     if(LedgerFinal.size()>=4){
197                                     if(LedgerFinal.get(LedgerFinal.size()-4)!=null);{
198                                             Label4.setText(LedgerFinal.get(LedgerFinal.size()-4));
199                                     }
200                                     }
201
202                                     if(LedgerFinal.size()>=5){
203                                     if(LedgerFinal.get(LedgerFinal.size()-5)!=null);{
204                                             Label5.setText(LedgerFinal.get(LedgerFinal.size()-5));
205                                     }
206                                     }
207
208                                     if(LedgerFinal.size()>=6){
209                                     if(LedgerFinal.get(LedgerFinal.size()-6)!=null);{
210                                             Label6.setText(LedgerFinal.get(LedgerFinal.size()-6));
211                                     }
212                                     }
213
214                                     if(LedgerFinal.size()>=7){
215                                     if(LedgerFinal.get(LedgerFinal.size()-7)!=null);{
216                                             Label7.setText(LedgerFinal.get(LedgerFinal.size()-7));
217                                     }
218                                     }
219
220                                     if(LedgerFinal.size()>=8){
221                                     if(LedgerFinal.get(LedgerFinal.size()-8)!=null);{
222                                             Label8.setText(LedgerFinal.get(LedgerFinal.size()-8));
223                                     }
224                                     }
225
226                                     if(LedgerFinal.size()>=9){
227                                     if(LedgerFinal.get(LedgerFinal.size()-9)!=null);{
228                                             Label9.setText(LedgerFinal.get(LedgerFinal.size()-9));
229                                     }
230                                     }
231
232                                     if(LedgerFinal.size()>=10){
233                                     if(LedgerFinal.get(LedgerFinal.size()-10)!=null);{
234                                             Label10.setText(LedgerFinal.get(LedgerFinal.size()-10));
235                                     }
236                                     }
237
238                                     if(LedgerFinal.size()>=11){
239                                     if(LedgerFinal.get(LedgerFinal.size()-11)!=null);{
240                                             Label11.setText(LedgerFinal.get(LedgerFinal.size()-11));
241                                     }
242                                     }
243
244                                     if(LedgerFinal.size()>=12){
245                                     if(LedgerFinal.get(LedgerFinal.size()-12)!=null);{
246                                             Label12.setText(LedgerFinal.get(LedgerFinal.size()-12));
247                                     }
248                                     }
249
250                                     if(LedgerFinal.size()>=13){
251                                     if(LedgerFinal.get(LedgerFinal.size()-13)!=null);{
252                                             Label13.setText(LedgerFinal.get(LedgerFinal.size()-13));
253                                     }
```

```
                                        }

                                        if(LedgerFinal.size()>=14){
                                        if(LedgerFinal.get(LedgerFinal.size()-14)!=null);{
                                                Label14.setText(LedgerFinal.get(LedgerFinal.size()-14));
                                        }
                                        }

                                        if(LedgerFinal.size()>=15){
                                        if(LedgerFinal.get(LedgerFinal.size()-15)!=null);{
                                                Label15.setText(LedgerFinal.get(LedgerFinal.size()-15));
                                        }
                                        }
                        }
                };

                timer.scheduleAtFixedRate(task, 0, 50);




        }

        public static void main(String[] args) {
                if (args.length < 2) {
                        System.out.println("Usage: demo.blockChain.BServer <server id> <ip administrator>");
                        System.exit(-1);
                }
                JFrame frame = new JFrame("    Server");
                frame.setContentPane(new BServer<String,
    String>(Integer.parseInt(args[0]),args[1],frame).contentPane);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);


        }

        @SuppressWarnings("unchecked")
        @Override
        public byte[] appExecuteOrdered(byte[] command, MessageContext msgCtx) {
                byte[] reply = null;
                K key = null;
                V value = null;
                boolean hasReply = false;
                System.out.print("The ledger has the following records :");

                for (int i=0; i<LedgerFinal.size(); i=i+1){
                        System.out.print(" "+LedgerFinal.get(i));
                        }

                System.out.println();
                try (ByteArrayInputStream byteIn = new ByteArrayInputStream(command);
                                ObjectInput objIn = new ObjectInputStream(byteIn);
                                ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
                                ObjectOutput objOut = new ObjectOutputStream(byteOut);) {

                        BRequestType reqType = (BRequestType)objIn.readObject();
                        switch (reqType) {
                                case APPENDLEDGER:

                                        key = (K)objIn.readObject();
                                        System.out.println("The APPEND message with C="+key+" has been delivered");
                                        value = (V)objIn.readObject();
                                        String valuetemp=(String)value;
                                        LedgerFinal.add(valuetemp);

                                        System.out.println(LedgerFinal.size());
                                        try {
                                                FileWriter myWriter = new FileWriter(myObj,true);
                                                String tmpValueWrite=(String)value;
                                                myWriter.append(tmpValueWrite+"\n");
                                                myWriter.close();
                                                System.out.println("Successfully wrote to the file.");
                                                int Portnum=8080+PortId;
                                                Socket socket = new Socket(IpAddAdmin, Portnum);
                                                DataOutputStream outputSock = new
    DataOutputStream(socket.getOutputStream());
                                                outputSock.writeBytes(tmpValueWrite);
                                                socket.close();
```

```
                                        }
                                        catch (IOException e) {
                                            System.out.println("An error occurred.");
                                            e.printStackTrace();
                                            }


                                        String Response1 = "ACK";
                                        System.out.println("The "+Response1+" send to the client");
                                        if (Response1 != null) {
                                                objOut.writeObject(Response1);
                                                hasReply = true;
                                        }
                                        break;
                                case GETLEDGER:

                                        key = (K)objIn.readObject();
                                        System.out.println("The GET message with C="+key+" has been delivered");
                                        getset(objOut);
                                        hasReply = true;


                                        break;

                        }
                        if (hasReply) {
                                objOut.flush();
                                byteOut.flush();
                                reply = byteOut.toByteArray();
                        } else {
                                reply = new byte[0];
                        }

                } catch (IOException | ClassNotFoundException e) {
                        logger.log(Level.SEVERE, "Ocurred during map operation execution", e);
                }
                return reply;
        }

        @SuppressWarnings("unchecked")
        @Override
        public byte[] appExecuteUnordered(byte[] command, MessageContext msgCtx) {
                byte[] reply = null;
                K key = null;
                V value = null;
                boolean hasReply = false;


                System.out.println();
                try (ByteArrayInputStream byteIn = new ByteArrayInputStream(command);
                                ObjectInput objIn = new ObjectInputStream(byteIn);
                                ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
                                ObjectOutput objOut = new ObjectOutputStream(byteOut);) {
                        BRequestType reqType = (BRequestType)objIn.readObject();
                        switch (reqType) {
                                case GETLEDGER:

                                        key = (K)objIn.readObject();
                                        System.out.println("The GET message with C="+key+" has been delivered");
                                        getset(objOut);
                                        hasReply = true;


                                        break;
                                default:
                                        logger.log(Level.WARNING, "in appExecuteUnordered only read operations are
    supported");
                        }
                        if (hasReply) {
                                objOut.flush();
                                byteOut.flush();
                                reply = byteOut.toByteArray();
                        } else {
                                reply = new byte[0];
                        }
                } catch (IOException | ClassNotFoundException e) {
                        logger.log(Level.SEVERE, "Ocurred during map operation execution", e);
```

```
408                     logger.log(Level.SEVERE, "Ocurred during map operation execution", e);
409             }
410
411             return reply;
412     }
413
414
415
416     private void getset(ObjectOutput out) throws IOException, ClassNotFoundException {
417
418             int size = LedgerFinal.size();
419             out.writeInt(size);
420             for (int i=0; i<size; i=i+1){
421                     out.writeObject(LedgerFinal.get(i));
422             }
423
424     }
425
426
427
428     @Override
429     public byte[] getSnapshot() {
430             try (ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
431                             ObjectOutput objOut = new ObjectOutputStream(byteOut)) {
432                     objOut.writeObject(LedgerFinal);
433                     return byteOut.toByteArray();
434             } catch (IOException e) {
435                     logger.log(Level.SEVERE, "Error while taking snapshot", e);
436             }
437             return new byte[0];
438     }
439
440     @SuppressWarnings("unchecked")
441     @Override
442     public void installSnapshot(byte[] state) {
443             try (ByteArrayInputStrean byteIn = new ByteArrayInputStream(state);
444                             ObjectInput objIn = new ObjectInputStream(byteIn)) {
445                     LedgerFinal = (List<String>)objIn.readObject();
446             } catch (IOException | ClassNotFoundException e) {
447                     logger.log(Level.SEVERE, "Error while installing snapshot", e);
448             }
449     }
450 }
```

Παράρτημα Β

```java
package bftsmart.demo.blockchain;
import java.io.File;
import java.io.FileWriter;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;
import java.util.Map;
import java.util.Set;
import java.util.ArrayList;
import java.util.List;
import java.util.TreeMap;
import java.util.TreeSet;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import java.awt.Color;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import java.util.ArrayList;
import java.util.List;
import java.awt.Font;
import java.util.Timer;
import java.util.TimerTask;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import java.net.Socket;
import java.util.Date;
import java.util.Scanner;

import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.util.List;
import java.awt.event.ActionEvent;

import bftsmart.tom.MessageContext;
import bftsmart.tom.ServiceReplica;
import bftsmart.tom.server.defaultservices.DefaultSingleRecoverable;

public class BFServer<K, V> extends DefaultSingleRecoverable {


        private Logger logger;
        private String FileName;
        private int    PortId;
        private String IpAddAdmin;
        private File myObj;
        private List<String> LedgerFinal;
        private JPanel contentPane;
        private JLabel Label1;
        private JLabel Label2;
        private JLabel Label3;
        private JLabel Label4;
        private JLabel Label5;
        private JLabel Label6;
        private JLabel Label7;
        private JLabel Label8;
        private JLabel Label9;
        private JLabel Label10;
        private JLabel Label11;
        private JLabel Label12;
        private JLabel Label13;
        private JLabel Label14;
        private JLabel Label15;

        public BFServer(int id,String IpAddressAdmin,JFrame frame) {
                FileName="LogFile"+id+"Server.txt";
                PortId=id;
                IpAddAdmin= IpAddressAdmin;
                try {
                        myObj = new File(FileName);
                        if (myObj.createNewFile()) {
                                System.out.println("File created: " + myObj.getName());
                        } else {
                                System.out.println("File already exists.");
                        }
                }
                catch (IOException e) {
                        System.out.println("An error occurred.");
                        e.printStackTrace();
```

```
87              }
88
89              System.out.println("The ip of the administrator is " + IpAddAdmin);
90              LedgerFinal = new ArrayList<String>();
91              logger = Logger.getLogger(BServer.class.getName());
92              new ServiceReplica(id, this, this);
93
94              frame.setBackground(Color.GRAY);
95              frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
96              frame.setBounds(100, 100, 649, 850);
97              contentPane = new JPanel();
98              contentPane.setBackground(Color.GRAY);
99              contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
100             frame.setContentPane(contentPane);
101             contentPane.setLayout(null);
102
103             JLabel lblNewLabel = new JLabel("Server # ");
104             lblNewLabel.setText("Server"+id);
105             lblNewLabel.setFont(new Font("Tahoma", Font.BOLD, 15));
106             lblNewLabel.setBounds(269, 27, 112, 44);
107             contentPane.add(lblNewLabel);
108
109             JLabel Label1 = new JLabel("");
110             Label1.setBounds(33, 105, 200, 16);
111             contentPane.add(Label1);
112
113             JLabel Label2 = new JLabel("");
114             Label2.setBounds(33, 144, 200, 16);
115             contentPane.add(Label2);
116
117             JLabel Label3 = new JLabel("");
118             Label3.setBounds(33, 183, 200, 16);
119             contentPane.add(Label3);
120
121             JLabel Label4 = new JLabel("");
122             Label4.setBounds(33, 224, 200, 16);
123             contentPane.add(Label4);
124
125             JLabel Label5 = new JLabel("");
126             Label5.setBounds(33, 264, 200, 16);
127             contentPane.add(Label5);
128
129             JLabel Label6 = new JLabel("");
130             Label6.setBounds(33, 306, 200, 16);
131             contentPane.add(Label6);
132
133             JLabel Label7 = new JLabel("");
134             Label7.setBounds(33, 345, 200, 16);
135             contentPane.add(Label7);
136
137             JLabel Label8 = new JLabel("");
138             Label8.setBounds(33, 388, 200, 16);
139             contentPane.add(Label8);
140
141             JLabel Label9 = new JLabel("");
142             Label9.setBounds(33, 428, 200, 16);
143             contentPane.add(Label9);
144
145             JLabel Label10 = new JLabel("");
146             Label10.setBounds(33, 468, 200, 16);
147             contentPane.add(Label10);
148
149             JLabel Label11 = new JLabel("");
150             Label11.setBounds(33, 507, 200, 16);
151             contentPane.add(Label11);
152
153             JLabel Label12 = new JLabel("");
154             Label12.setBounds(33, 548, 200, 16);
155             contentPane.add(Label12);
156
157             JLabel Label13 = new JLabel("");
158             Label13.setBounds(33, 587, 200, 16);
159             contentPane.add(Label13);
160
161             JLabel Label14 = new JLabel("");
162             Label14.setBounds(33, 629, 200, 16);
163             contentPane.add(Label14);
164
165             JLabel Label15 = new JLabel("");
```

```
166             Label15.setBounds(33, 670, 200, 16);
167             contentPane.add(Label15);
168
169             Timer timer = new Timer();
170             TimerTask task = new TimerTask() {
171                     int counterian=0;
172                     @Override
173                     public void run() {
174                                     if(LedgerFinal.size()>=1){
175                                     if(LedgerFinal.get(LedgerFinal.size()-1)!=null);{
176                                             Label1.setText(LedgerFinal.get(LedgerFinal.size()-1));
177
178                                     }
179                                     }
180
181                                     if(LedgerFinal.size()>=2){
182                                     if(LedgerFinal.get(LedgerFinal.size()-2)!=null);{
183                                             Label2.setText(LedgerFinal.get(LedgerFinal.size()-2));
184                                     }
185                                     }
186
187                                     if(LedgerFinal.size()>=3){
188                                     if(LedgerFinal.get(LedgerFinal.size()-3)!=null);{
189                                             Label3.setText(LedgerFinal.get(LedgerFinal.size()-3));
190                                     }
191                                     }
192
193                                     if(LedgerFinal.size()>=4){
194                                     if(LedgerFinal.get(LedgerFinal.size()-4)!=null);{
195                                             Label4.setText(LedgerFinal.get(LedgerFinal.size()-4));
196                                     }
197                                     }
198
199                                     if(LedgerFinal.size()>=5){
200                                     if(LedgerFinal.get(LedgerFinal.size()-5)!=null);{
201                                             Label5.setText(LedgerFinal.get(LedgerFinal.size()-5));
202                                     }
203                                     }
204
205                                     if(LedgerFinal.size()>=6){
206                                     if(LedgerFinal.get(LedgerFinal.size()-6)!=null);{
207                                             Label6.setText(LedgerFinal.get(LedgerFinal.size()-6));
208                                     }
209                                     }
210
211                                     if(LedgerFinal.size()>=7){
212                                     if(LedgerFinal.get(LedgerFinal.size()-7)!=null);{
213                                             Label7.setText(LedgerFinal.get(LedgerFinal.size()-7));
214                                     }
215                                     }
216
217                                     if(LedgerFinal.size()>=8){
218                                     if(LedgerFinal.get(LedgerFinal.size()-8)!=null);{
219                                             Label8.setText(LedgerFinal.get(LedgerFinal.size()-8));
220                                     }
221                                     }
222
223                                     if(LedgerFinal.size()>=9){
224                                     if(LedgerFinal.get(LedgerFinal.size()-9)!=null);{
225                                             Label9.setText(LedgerFinal.get(LedgerFinal.size()-9));
226                                     }
227                                     }
228
229                                     if(LedgerFinal.size()>=10){
230                                     if(LedgerFinal.get(LedgerFinal.size()-10)!=null);{
231                                             Label10.setText(LedgerFinal.get(LedgerFinal.size()-10));
232                                     }
233                                     }
234
235                                     if(LedgerFinal.size()>=11){
236                                     if(LedgerFinal.get(LedgerFinal.size()-11)!=null);{
237                                             Label11.setText(LedgerFinal.get(LedgerFinal.size()-11));
238                                     }
239                                     }
240
241                                     if(LedgerFinal.size()>=12){
242                                     if(LedgerFinal.get(LedgerFinal.size()-12)!=null);{
243                                             Label12.setText(LedgerFinal.get(LedgerFinal.size()-12));
244                                     }
245                                     }
246
```

```
247                                     if(LedgerFinal.size()>=13){
248                                     if(LedgerFinal.get(LedgerFinal.size()-13)!=null);{
249                                             Label13.setText(LedgerFinal.get(LedgerFinal.size()-13));
250                                     }
251                                     }
252
253                                     if(LedgerFinal.size()>=14){
254                                     if(LedgerFinal.get(LedgerFinal.size()-14)!=null);{
255                                             Label14.setText(LedgerFinal.get(LedgerFinal.size()-14));
256                                     }
257                                     }
258
259                                     if(LedgerFinal.size()>=15){
260                                     if(LedgerFinal.get(LedgerFinal.size()-15)!=null);{
261                                             Label15.setText(LedgerFinal.get(LedgerFinal.size()-15));
262                                     }
263                                     }
264                             }
265                     };
266
267                     timer.scheduleAtFixedRate(task, 0, 50);
268
269
270
271
272             }
273
274             public static void main(String[] args) {
275                     if (args.length < 2) {
276                             System.out.println("Usage: demo.blockChain.BServer <server id> <ip administrator>");
277                             System.exit(-1);
278                     }
279                     JFrame frame = new JFrame("    Server");
280                     frame.setContentPane(new BFServer<String,
    String>(Integer.parseInt(args[0]),args[1],frame).contentPane);
281                     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
282                     frame.setVisible(true);
283
284
285             }
286
287             @SuppressWarnings("unchecked")
288             @Override
289             public byte[] appExecuteOrdered(byte[] command, MessageContext msgCtx) {
290                     byte[] reply = null;
291                     K key = null;
292                     V value = null;
293                     boolean hasReply = false;
294                     System.out.print("The ledger has the following records :");
295
296                     for (int i=0; i<LedgerFinal.size(); i=i+1){
297                             System.out.print(" "+LedgerFinal.get(i));
298                             }
299
300                     System.out.println();
301                     try (ByteArrayInputStream byteIn = new ByteArrayInputStream(command);
302                                     ObjectInput objIn = new ObjectInputStream(byteIn);
303                                     ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
304                                     ObjectOutput objOut = new ObjectOutputStream(byteOut);) {
305
306                             BRequestType reqType = (BRequestType)objIn.readObject();
307                             switch (reqType) {
308                                     case APPENDLEDGER:
309
310                                             key = (K)objIn.readObject();
311                                             System.out.println("The APPEND message with C="+key+" has been delivered");
312                                             value = (V)objIn.readObject();
313                                             String valuetemp="Im faulty";
314                                             LedgerFinal.add(valuetemp);
315
316                                             System.out.println(LedgerFinal.size());
317                                             try {
318                                                     FileWriter myWriter = new FileWriter(myObj,true);
319                                                     String tmpValueWrite="Im faulty";
320                                                     myWriter.append(tmpValueWrite+"\n");
321                                                     myWriter.close();
322                                                     System.out.println("Successfully wrote to the file.");
323                                                     int Portnum=8080+PortId;
324                                                     Socket socket = new Socket(IpAddAdmin, Portnum);
325                                                     DataOutputStream outputSock = new
    DataOutputStream(socket.getOutputStream());
326                                                     outputSock.writeBytes(tmpValueWrite);
```

```
327                                         socket.close();
328
329                                     }
330                                     catch (IOException e) {
331                                         System.out.println("An error occurred.");
332                                         e.printStackTrace();
333                                         }
334
335
336                                     String Response1 = "ACK";
337                                     System.out.println("The "+Response1+" send to the client");
338                                     if (Response1 != null) {
339                                             objOut.writeObject(Response1);
340                                             hasReply = true;
341                                     }
342                                     break;
343                             case GETLEDGER:
344
345                                     key = (K)objIn.readObject();
346                                     System.out.println("The GET message with C="+key+" has been delivered");
347                                     getset(objOut);
348                                     hasReply = true;
349
350
351                                     break;
352
353                     }
354                     if (hasReply) {
355                             objOut.flush();
356                             byteOut.flush();
357                             reply = byteOut.toByteArray();
358                     } else {
359                             reply = new byte[0];
360                     }
361
362             } catch (IOException | ClassNotFoundException e) {
363                     logger.log(Level.SEVERE, "Ocurred during map operation execution", e);
364             }
365             return reply;
366     }
367
368     @SuppressWarnings("unchecked")
369     @Override
370     public byte[] appExecuteUnordered(byte[] command, MessageContext msgCtx) {
371             byte[] reply = null;
372             K key = null;
373             V value = null;
374             boolean hasReply = false;
375
376
377
378             System.out.println();
379             try (ByteArrayInputStrean byteIn = new ByteArrayInputStream(command);
380                             ObjectInput objIn = new ObjectInputStream(byteIn);
381                             ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
382                             ObjectOutput objOut = new ObjectOutputStream(byteOut);) {
383                     BRequestType reqType = (BRequestType)objIn.readObject();
384                     switch (reqType) {
385                             case GETLEDGER:
386
387                                     key = (K)objIn.readObject();
388                                     System.out.println("The GET message with C="+key+" has been delivered");
389                                     getset(objOut);
390                                     hasReply = true;
391
392
393
394                                     break;
395                             default:
396                                     logger.log(Level.WARNING, "in appExecuteUnordered only read operations are
    supported");
397                     }
398                     if (hasReply) {
399                             objOut.flush();
400                             byteOut.flush();
401                             reply = byteOut.toByteArray();
402                     } else {
403                             reply = new byte[0];
404                     }
405             } catch (IOException | ClassNotFoundException e) {
406                     logger.log(Level.SEVERE, "Ocurred during map operation execution", e);
407             }
408
```

```
409         }
410
411
412
413         private void getset(ObjectOutput out) throws IOException, ClassNotFoundException {
414
415                 int size = LedgerFinal.size();
416                 out.writeInt(size);
417                 for (int i=0; i<size; i=i+1){
418                         out.writeObject(LedgerFinal.get(i));
419                 }
420
421         }
422         }
423
424
425
426         @Override
427         public byte[] getSnapshot() {
428                 try (ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
429                                 ObjectOutput objOut = new ObjectOutputStream(byteOut)) {
430                         objOut.writeObject(LedgerFinal);
431                         return byteOut.toByteArray();
432                 } catch (IOException e) {
433                         logger.log(Level.SEVERE, "Error while taking snapshot", e);
434                 }
435                 return new byte[0];
436         }
437
438         @SuppressWarnings("unchecked")
439         @Override
440         public void installSnapshot(byte[] state) {
441                 try (ByteArrayInputStream byteIn = new ByteArrayInputStream(state);
442                                 ObjectInput objIn = new ObjectInputStream(byteIn)) {
443                         LedgerFinal = (List<String>)objIn.readObject();
444                 } catch (IOException | ClassNotFoundException e) {
445                         logger.log(Level.SEVERE, "Error while installing snapshot", e);
446                 }
447         }
448 }
```

Παράρτημα Γ

```java
package bftsmart.demo.blockchain;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Date;
import java.util.ArrayList;
import java.util.List;
import java.io.File;
import java.io.FileWriter;

public class SingleThreadedTCPServer {

    public static void main(String args[]) {
        try {
                int ServerId=Integer.parseInt(args[0]);
                int PortNum=8080+ServerId;
                String clientbuffer = "";

                ServerSocket socket = new ServerSocket(PortNum);
                System.out.println("Server listening to: " + socket.getInetAddress() + ":" + socket.getLocalPort());
                List<String> TempLedger = new ArrayList<String>();
                while (true) {
                        Socket client = socket.accept();
                        System.out.println("Client connected with: " + client.getInetAddress());
                        BufferedReader reader = new BufferedReader(
                                new InputStreamReader(client.getInputStream())
                        );
                clientbuffer = reader.readLine() + System.lineSeparator();
                TempLedger.add(clientbuffer);
                System.out.println("I receive from client "+ServerId+" the data : " + clientbuffer);
                System.out.println("The contennts of the Ledger is ");
                String FileName="LogFile"+ServerId+".txt";
                 File myObj = new File(FileName);
                 try {

                        if (myObj.createNewFile()) {
                                System.out.println("File created: " + myObj.getName());
                        } else {
                                System.out.println("File already exists.");
                        }
                 }
                 catch (IOException e) {
                        System.out.println("An error occurred.");
                        e.printStackTrace();
                 }


                try {
                        FileWriter myWriter = new FileWriter(myObj);
                        for(int i=TempLedger.size()-1; i>=0; i=i-1){
                                System.out.println(TempLedger.get(i));
                                myWriter.write(TempLedger.get(i));

                        }

                        myWriter.close();
                        System.out.println("Successfully wrote to the file.");
                 }

                 catch (IOException e) {
                        System.out.println("An error occurred.");
                        e.printStackTrace();
                 }


        }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

```
                                int size = objIn.readInt();
                                List<String> result = new ArrayList<String>();
                                while (size-- > 0) {
                                        result.add((String)objIn.readObject());
                                }
                                return result;
                        }

                } catch (IOException | ClassNotFoundException e) {
                        System.out.println("Exception getting keyset from map: " + e.getMessage());
                }
                return null;
        }


        @SuppressWarnings("unchecked")
        @Override
        public V remove(Object key) {
                throw new UnsupportedOperationException("Not supported yet.");
        }
        public BClient(int clientId) {
                serviceProxy = new ServiceProxy(clientId);
        }

        @SuppressWarnings("unchecked")
        @Override
        public V put(K key, V value) {
                throw new UnsupportedOperationException("Not supported yet.");
        }

        @SuppressWarnings("unchecked")
        @Override
        public V get(Object key) {
                throw new UnsupportedOperationException("Not supported yet.");
        }
        @Override
        public int size() {
                throw new UnsupportedOperationException("Not supported yet.");
        }

        @SuppressWarnings("unchecked")
        public Set<K> keySet(Object key) {
                throw new UnsupportedOperationException("Not supported yet.");
        }




        public void close() {
                serviceProxy.close();
        }

        @Override
        public void clear() {
                throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public boolean containsKey(Object key) {
                throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public boolean containsValue(Object value) {
                throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public Set<Entry<K, V>> entrySet() {
                throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public boolean isEmpty() {
                throw new UnsupportedOperationException("Not supported yet.");
        }
        @Override
        public Set<K> keySet() {
                throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public void putAll(Map<? extends K, ? extends V> m) {
                throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public Collection<V> values() {
                throw new UnsupportedOperationException("Not supported yet.");
        }
}
```

Παράρτημα Δ

```java
package bftsmart.demo.blockchain;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;
import java.util.Collection;
import java.util.ArrayList;
import java.util.List;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;


import bftsmart.tom.ServiceProxy;

public class BClient<K, V> implements Map<K, V>{

	ServiceProxy serviceProxy;



	@SuppressWarnings("unchecked")

	public V AppendLedger(K key, V value) {
		try (ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
				ObjectOutput objOut = new ObjectOutputStream(byteOut);) {

			objOut.writeObject(BRequestType.APPENDLEDGER);
			objOut.writeObject(key);
			objOut.writeObject(value);

			objOut.flush();
			byteOut.flush();
			System.out.println("The message APPEND with C="+key+" has been broadcasted to the servers of the Ledger");

			byte[] reply = serviceProxy.invokeOrdered(byteOut.toByteArray());
			System.out.println("The same reply from at least f+1 different servers for the "+key+" request have just arrived");
			if (reply.length == 0)
				return null;
			try (ByteArrayInputStream byteIn = new ByteArrayInputStream(reply);
					ObjectInput objIn = new ObjectInputStream(byteIn)) {

				return (V)objIn.readObject();
			}

		} catch (IOException | ClassNotFoundException e) {
			System.out.println("Exception putting value into map: " + e.getMessage());
		}
		return null;
	}

	@SuppressWarnings("unchecked")
	public List<String> GetLedger(Object key) {
		try (ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
				ObjectOutput objOut = new ObjectOutputStream(byteOut);) {

			objOut.writeObject(BRequestType.GETLEDGER);
			objOut.writeObject(key);

			objOut.flush();
			byteOut.flush();
			System.out.println("The message GET with C="+key+" has been broadcasted to the servers of the Ledger");

			byte[] reply = serviceProxy.invokeOrdered(byteOut.toByteArray());
			try (ByteArrayInputStream byteIn = new ByteArrayInputStream(reply);
					ObjectInput objIn = new ObjectInputStream(byteIn)) {
				System.out.println("The same reply from at least f+1 different servers for the "+key+" request have just arrived");
				int size = objIn.readInt();
				List<String> result = new ArrayList<String>();
				while (size-- > 0) {
					result.add((String)objIn.readObject());
				}
```

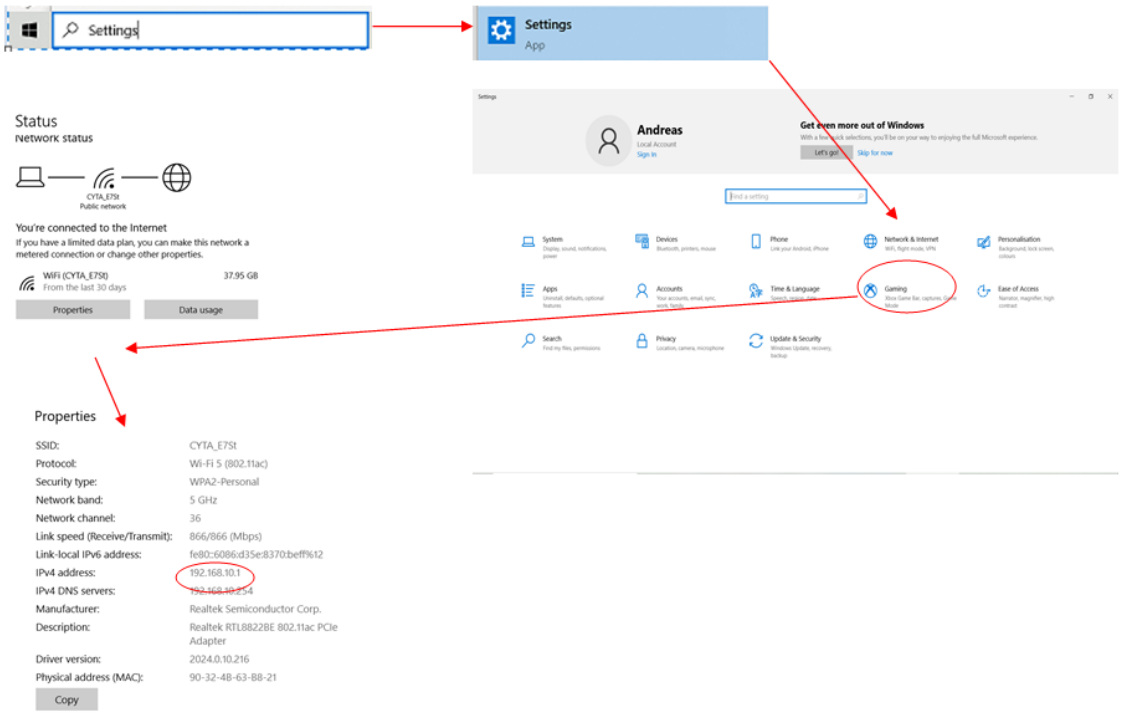
Παράρτημα Ε

Στο παραδοτέο υπάρχουν 2 φάκελοι ο library και ο BDLO1. Ο library περιέχει ότι χρησιμοποιήθηκε για την δημιουργία του συστήματος, όπως τον πηγαίο κώδικα, παραδείγματα που ήρθαν μαζί με την βιβλιοθήκη κλπ. Ο δεύτερος φάκελος είναι αυτός που πρέπει να είναι εγκατεστημένος στης μηχανές που θα χρησιμοποιούν το σύστημα, και περιέχει το φάκελο lib στον οποίο είναι αποθηκευμένα όλα τα Jar αρχεία, περιέχει τα configuration file και τα scripts που χρησιμοποιούμε για να εκτελεσθούν οι κλάσεις.

Για να ξεκινήσει η διαδικασία πρέπει να διαμορφωθούν σωστά τα 2 configuration file.

Για το host.config κάνουμε την εξής διαδικασία:

1)Βρίσκουμε την IP διεύθυνση των μηχανών στις οποίες θα τρέχουν οι εξυπηρετητές (Στο Windows Search πληκτρολογούμε Settings → Network&Internet → Properties και είναι η διεύθυνση που βρίσκεται διπλά από το IPV4 address:)



2)Ενημερώνουμε το αρχείο hosts.cofig (βρίσκεται στο φάκελο config που βρίσκεται στο φάκελο BDLO1) με τις IP διευθύνσεις κάθε μηχανής, τη πόρτα που θέλουμε να χρησιμοποίει για την επικοινωνία με πελάτες και την πόρτα που θέλουμε να χρησιμοποιεί για την επικοινωνία με τους εξυπηρετητές, αυτές οι πόρτες πρέπει να είναι διαφορετικές. Προσοχή όταν τρέχει κάποιος εξυπηρετητής έχει σαν παράμετρο ένα id. Στο αρχείο hosts,config η ταυτότητα του εξυπηρετητή μπαίνει δίπλα από την διεύθυνση IP στην οποία τρέχει, δίπλα από την διεύθυνση IP είναι η πόρτα που επικοινωνεί ο εξυπηρετητής με τους πελάτες και δίπλα η πόρτα που επικοινωνεί με τους εξυπηρετητές.

```
0 127.0.0.1 11000 11001
1 127.0.0.1 11010 11011
2 127.0.0.1 11020 11021
3 127.0.0.1 11030 11031
4 127.0.0.1 11040 11041
```

Σχήμα Ε.1: Το hosts.config αρχείο

3) Τώρα πρέπει να ενημερώσουμε το αρχείο system.config με τις τιμές των απαραίτητων παραμέτρων ώστε να διαμορφωθεί σωστά το σύστημα. Αρχικά στην παράμετρο system.servers.num δηλώνουμε τον αριθμό των εξυπηρετητών που αποτελούν το σύστημα μας, στην παράμετρο system.servers.f δηλώνουμε τον αριθμό των Βυζαντινών εξυπηρετητών, στην παράμετρο, system.initial.view τοποθετούμε τις ταυτότητες των εξυπηρετητών (π.χ. 0,1,2,3 για 4 εξυπηρετητές), εδώ προσοχή οι ταυτότητες των εξυπηρετητών πρέπει να ξεκινούν από το 0 και να αυξάνονται κατά 1 κάθε φορά. Επίσης πρέπει η παράμετρος system.bft να έχει τιμή true, αυτό δηλώνει ότι θέλουμε να το τρέξουμε για την Βυζαντινή έκδοση, και τέλος στην παράμετρο system.communication.bindaddress τοποθετούμε την IP διεύθυνση όπως την βρήκαμε στο βήμα 1 (αυτό ισχύει μόνο στην περιπτώσεις που θα τρέξουμε εξυπηρετητή σε αυτό το μηχάνημα)

4) Τώρα που έγινε η διαμόρφωση του συστήματος είμαστε έτοιμοι να το τρέξουμε, αρχικά πρέπει να αρχικοποιηθούν όλοι οι εξυπηρετητές. Αυτό επιτυγχάνεται μέσω της γραμμής εντολής, πρέπει να περιηγηθούμε μέσω της γραμμής εντολής και να μπούμε στο φάκελο BDLO1, από εκεί εκτελούμε την εντολή:
./runscripts/smartrun.sh bftsmart/demo/blockchain/BServer **ID IP_OfAdministrator**
για Linux
Και την εντολή:
runscripts\smartrun.cmd bftsmart/demo/blockchain/BServer **ID IP_OfAdministrator**
για Windows

5) Τώρα που αρχικοποιήθηκαν οι εξυπηρετητές θα αρχικοποιηθούν και οι απαραίτητες κλάσεις από πλευράς διαχειριστή. Αρχικά θα πρέπει να αρχικοποιηθούν οι κλάσεις που επικοινωνούν με τους εξυπηρετητές (1 instance για κάθε εξυπηρετητή και πρέπει να έχουν το ίδιο id με αυτό του εξυπηρετητή που επικοινωνούν) για την δημιουργία του αρχείου που διαβάζει ο διαχειριστής.
Για Unix γίνεται με την ακόλουθη εντολή:
./runscripts/smartrun.sh bftsmart/demo/blockchain/SingleThreadedTCPServer **ID_Server**
Και για Windows με την ακόλουθη εντολή:
runscripts\smartrun.cmd bftsmart/demo/blockchain/SingleThreadedTCPServer **ID_Server**

Ακολούθως θα αρχικοποιηθεί και ο διαχειριστής ο οποίος αρχικοποιείται στα Unix με την ακόλουθη εντολή:
./runscripts/smartrun.sh bftsmart/demo/blockchain/ServerAdministrator5Server **NumberOfServers**
Και στα Windows με την ακόλουθη εντολή:
runscripts\smartrun.cmd bftsmart/demo/blockchain/ServerAdministrator5Server **NumberOfServers**

Τώρα όλα είναι έτοιμα για να αρχικοποιηθούν οι πελάτες και να ξεκινήσουν να πραγματοποιούν αιτήματα. Όπως και οι εξυπηρετητές έτσι και οι πελάτες έχουν

αναγνωριστικό, έτσι κάθε πελάτης πρέπει να έχει διαφορετικό ID. Η αρχικοποίηση των πελατών γίνεται με την ακόλουθη εντολή για Unix:

./runscripts/smartrun.sh bftsmart/demo/blockchain/ClientGui **ID_Of_Client**

Και για windows την εντολή:

runscripts\smartrun.cmd bftsmart/demo/blockchain/ClientGui **ID_Of_Client**