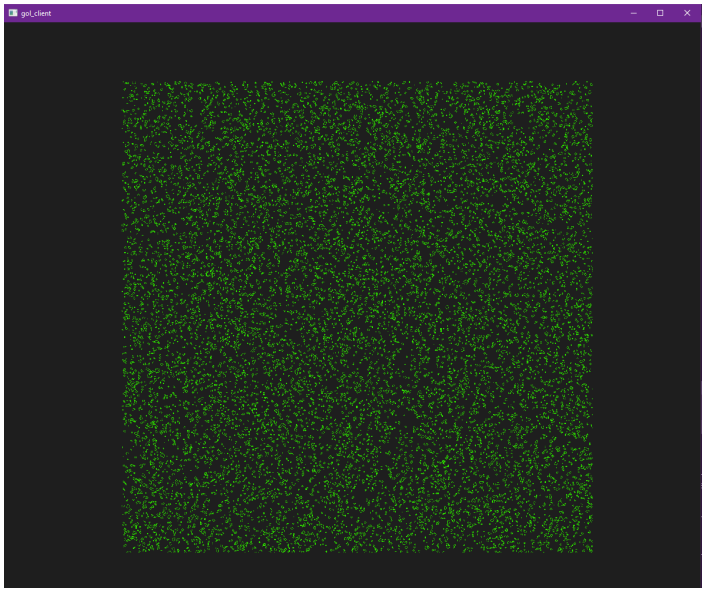
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Δημιουργία Πλατφόρμας Επιστημονικών Υπολογισμών μεγάλης κλίμακας»



Του φοιτητή
Ψειράκη Γεώργιου
Αρ. Μητρώου: 144168

Επιβλέπων
Καθηγητής
Στάθης Κασδερίδης

Ημερομηνία 07-11-2024

Τίτλος Δ.Ε. "Δημιουργία Πλατφόρμας Επιστημονικών Υπολογισμών μεγάλης κλίμακας"
Κωδικός Δ.Ε. 23191
Ονοματεπώνυμο φοιτητή : Ψειράκης Γεώργιος
Ονοματεπώνυμο εισηγητή : Στάθης Κασδερίδης
Ημερομηνία ανάληψης : Δ.Ε. 30-03-2023
Ημερομηνία περάτωσης : Δ.Ε. 07-11-2024

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«Στην οικογένειά μου»

Περίληψη

Η διπλωματική πραγματεύεται μια πλατφόρμα υψηλής απόδοσης υπολογισμών που χρειάζονται για την προσομοίωση επιστημονικών προβλημάτων μεγάλης κλίμακας. Συγκεκριμένα θα υποστηρίξει υπολογιστικά μοντέλα που έχουν υψηλή παραλληλία, όπως αυτά που εμφανίζονται σε προσομοιώσεις πρακτόρων, οικολογικών συστημάτων, περιβαλλοντικών φαινομένων και άλλα. Η κεντρική δομή δεδομένων θα είναι ένα πλέγμα θέσεων (GridSpace) που με την σειρά της θα κατανέμεται σε πολλαπλούς υπολογιστικούς κόμβους (cluster nodes) που ο καθένας από αυτούς θα εκτελεί τους υπολογισμούς που του αντιστοιχούν στο κομμάτι (slice) του πλέγματος που είναι υπεύθυνος. Γειτονικά κομμάτια του πλέγματος θα επικοινωνούν μεταξύ τους, μέσα από ένα μηχανισμό μηνυμάτων, για να ενημερώσουν την κατάσταση τους αφού πάρουν υπόψη αλλαγές μεταβλητών σε γείτονες που τα αφορούν. Η υλοποίηση χρησιμοποιεί γλώσσα Nim. Το μοντέλο χρήσης στηρίζεται στην κατανεμημένη δημιουργία/αλλαγή μεταβλητών σε κάθε κομμάτι του χώρου (slice) και γίνεται με τον ορισμό σχετικού API. Η διπλωματική για επαλήθευση της υλοποίησης χρησιμοποιεί το Game of Life μοντέλο.

«Development of a Large-Scale Scientific Computation Platform»

«George Pseirakis»

Abstract

The thesis implements a high-performance computing platform needed to simulate large-scale scientific problems. In particular, it will support computational models that are highly parallel, such as those seen in simulations of agents, ecological systems, environmental phenomena, and more. The central data structure will be a grid of cells (GridSpace) which will in turn be distributed to multiple computing nodes (cluster nodes) and each of them will perform the calculations corresponding to the slice of the grid for which it is responsible. Neighboring slices of the grid will communicate with each other, through a messaging mechanism, to update their status after considering variable changes in neighbors that concern them. The implementation uses the programming language Nim. The usage model is based on the distributed creation/change of variables in each slice of the space and is achieved by defining a relevant API. The thesis uses the Game of Life model for verification of the implementation.

Ευχαριστίες

Ευχαριστώ τους γονείς μου που με στήριξαν οικονομικά και ηθικά στην ολοκλήρωση της διπλωματικής εργασίας.

Ευχαριστώ επίσης τον επιβλέποντα καθηγητή κ. Στάθη Κασδερίδη, για τις κατευθύνσεις που μου έδωσε και την βοήθεια του σε προβλήματα που εμφανίστηκαν.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AST	Abstract Syntax Tree
CPU	Central Processing Unit
FFI	Foreign Function Interface
GB	Gigabyte
GOL	Game Of Life
GUI	Graphic User Interface
IoT	Internet of Things
JSON	JavaScript Object Notation
MPI	Message Passing Interface
MPICH	Message Passing Interface Chameleon
RAM	Random-Access Memory
RNG	Random Number Generator
SC	Scientific Computing
STOMP	Simple Text Orientated Messaging Protocol
UI	User Interface
VM	Virtual Machine

Εισαγωγή

Στο πρώτο κεφάλαιο θα εξερευνήσουμε την έννοια της επιστημονικής υπολογιστικής και της σημασίας της, πώς σχετίζεται με την παρούσα εργασία, και θα γίνει αναφορά σε τεχνολογίες αποστολής μηνυμάτων από μία εφαρμογή σε άλλη.

Στο δεύτερο κεφάλαιο θα παρουσιαστεί η αρχιτεκτονική του συστήματος, δηλαδή τα επιμέρους στοιχεία του και η αρμοδιότητά τους. Επίσης θα αναλυθεί η λειτουργικότητα του αλγορίθμου μέτρησης υπολογιστικής ισχύος, ο οποίος παίζει πολύ σημαντικό ρόλο στην ομαλή λειτουργία της εφαρμογής.

Στο τρίτο κεφάλαιο θα εξηγηθεί η εγκατάσταση των επιμέρους εφαρμογών που χρειάζονται για την δημιουργία και εκτέλεση της πλατφόρμας. Θα εξηγηθεί επίσης η υλοποίηση των κυρίων τμημάτων του κώδικα.

Το τέταρτο και τελευταίο κεφάλαιο είναι αυτό που θα επιδείξει την εφαρμογή στην πορεία εκτέλεσής της βήμα προς βήμα. Στο τέλος του κεφαλαίου συζητούνται προτάσεις βελτίωσης της εφαρμογής και αξιολόγηση των αποτελεσμάτων.

Κεφάλαιο 1ο: Ξεκινώντας

1.1 Λίγα λόγια για την Επιστημονική Υπολογιστική και τους στόχους της εργασίας

Η Επιστημονική Υπολογιστική είναι **η συλλογή εργαλείων, τεχνικών και θεωριών που απαιτούνται για την επίλυση μαθηματικών μοντέλων από προβλήματα στην Επιστήμη και τη Μηχανική σε υπολογιστή**. Η πλειονότητα αυτών των εργαλείων, τεχνικών και θεωριών αναπτύχθηκαν αρχικά στα Μαθηματικά και πολλά από αυτά είχαν τη γέννησή τους πολύ πριν από την εμφάνιση των ηλεκτρονικών υπολογιστών. Αυτό το σύνολο μαθηματικών θεωριών και τεχνικών ονομάζεται Αριθμητική Ανάλυση (ή Αριθμητικά Μαθηματικά) και αποτελεί σημαντικό μέρος της Επιστημονικής Υπολογιστικής. Η ανάπτυξη του ηλεκτρονικού υπολογιστή, ωστόσο, σηματοδότησε μια νέα εποχή στην προσέγγιση επίλυσης επιστημονικών προβλημάτων. Πολλές από τις αριθμητικές μεθόδους που είχαν αναπτυχθεί για υπολογισμούς με το χέρι (συμπεριλαμβανομένης της χρήσης επιτραπέζιων αριθμομηχανών) έπρεπε να αναθεωρηθούν και μερικές φορές να εγκαταλειφθούν. Θεωρήσεις που ήταν άσχετες ή ασήμαντες για τον υπολογισμό με το χέρι έγιναν πλέον υψίστης σημασίας για την αποτελεσματική και σωστή χρήση ενός μεγάλου υπολογιστικού συστήματος. Πολλές από αυτές τις θεωρήσεις – γλώσσες προγραμματισμού, λειτουργικά συστήματα, διαχείριση μεγάλων ποσοτήτων δεδομένων, ορθότητα προγραμμάτων – υπάγονται πλέον στη πειθαρχία της Επιστήμης των Υπολογιστών, από την οποία εξαρτάται πλέον σε μεγάλο βαθμό η Επιστημονική Υπολογιστική. Αλλά τα ίδια τα μαθηματικά συνεχίζουν να διαδραματίζουν σημαντικό ρόλο στον επιστημονική υπολογιστική. Παρέχουν τη γλώσσα των μαθηματικών μοντέλων που πρόκειται να λυθούν και πληροφορίες σχετικά με την καταλληλότητα ενός μοντέλου (Έχει λύση; Είναι η λύση μοναδική;) και παρέχουν την θεωρητική βάση για τις αριθμητικές μεθόδους.

Συνοπτικά, λοιπόν, η επιστημονική υπολογιστική βασίζεται στα μαθηματικά και την επιστήμη των υπολογιστών για να αναπτύξει τον καλύτερο τρόπο χρήσης συστημάτων υπολογιστών για την επίλυση προβλημάτων από την επιστήμη και τη μηχανική. [1]

Μία περίπτωση χρήσης επιστημονικής υπολογιστικής είναι για την πρόβλεψη καιρού. Οι επιστήμονες της ατμόσφαιρας και οι μετεωρολόγοι χρησιμοποιούν μοντέλα καιρού (μαθηματικές εξισώσεις) που εκτελούνται σε ισχυρούς υπολογιστές. Τα αποτελέσματα είναι γενικά ένας μεγάλος όγκος δεδομένων (θερμοκρασία, πίεση, υγρασία κ.λπ.) τα οποία συλλέγονται και απεικονίζονται σε χάρτες από διάφορες περιοχές. Τα δεδομένα που προκύπτουν δίνουν τις τυπικές προβλέψεις που πολλοί από εμάς βλέπουμε σε εφαρμογές Smartphone, στην τηλεόραση ή σε άλλες συσκευές.

Η ανάλυση δεδομένων είναι επίσης ένα πεδίο στο οποίο η επιστημονική υπολογιστική μπορεί να φανεί χρήσιμη. Ένα αυτόνομο όχημα για παράδειγμα μπορεί να παράγει 25 GB δεδομένων την ημέρα. Παρόλο που υπάρχει αφθονία επιστημονικών υπολογισμών χωρίς τη δημιουργία δεδομένων σε αυτοοδηγούμενα αυτοκίνητα, αυτό είναι ένα παράδειγμα της ανάγκης χειρισμού και ανάλυσης μεγάλου όγκου δεδομένων αρκετά γρήγορα. [2]

1.2 Η Επιστημονική Υπολογιστική στα πλαίσια της διπλωματικής

Η διπλωματική θα σχεδιάσει και θα υλοποιήσει μια πλατφόρμα υψηλής απόδοσης υπολογισμών που χρειάζονται για την προσομοίωση επιστημονικών προβλημάτων μεγάλης κλίμακας. Συγκεκριμένα θα υποστηρίξει υπολογιστικά μοντέλα που έχουν υψηλή παραλληλία, όπως αυτά που εμφανίζονται σε προσομοιώσεις πρακτόρων, οικολογικών συστημάτων, περιβαλλοντικών φαινομένων και άλλα. Η κεντρική δομή δεδομένων θα είναι ένα πλέγμα θέσεων (GridSpace) που με την σειρά της θα κατανέμεται σε πολλαπλούς υπολογιστικούς κόμβους (cluster nodes) που ο καθένας από αυτούς θα εκτελεί τους υπολογισμούς που του αντιστοιχούν στο κομμάτι (slice) του πλέγματος που είναι υπεύθυνος. Γειτονικά κομμάτια του πλέγματος θα επικοινωνούν μεταξύ τους, μέσα από ένα μηχανισμό μηνυμάτων, για να ενημερώσουν την κατάσταση τους αφού πάρουν υπόψη αλλαγές μεταβλητών σε γείτονες που τα αφορούν. Η υλοποίηση θα χρησιμοποιήσει γλώσσα Nim. Το μοντέλο χρήσης στηρίζεται στην κατανεμημένη δημιουργία/αλλαγή μεταβλητών σε κάθε κομμάτι του χώρου (slice) και γίνεται με τον ορισμό σχετικού API. Η διπλωματική για επαλήθευση της υλοποίησης θα υλοποιήσει το Game of Life.

Λόγω του τεράστιου όγκου δεδομένων που πρέπει να δέχεται η πλατφόρμα, οι συνηθισμένοι τρόποι και τεχνολογίες διαχείρισης, ανάλυσης και επεξεργασίας δεδομένων είναι ανεπαρκείς. Σε αυτή την εφαρμογή θα γίνει για αυτόν τον λόγο χρήση τεχνολογιών αποστολής μηνυμάτων μεταξύ υπολογιστικών συστημάτων σε ένα δίκτυο, και διαχωρισμός του συνόλου των δεδομένων σε (λογικές) φέτες (slices).

1.3 Τεχνολογίες αποστολής Μηνυμάτων

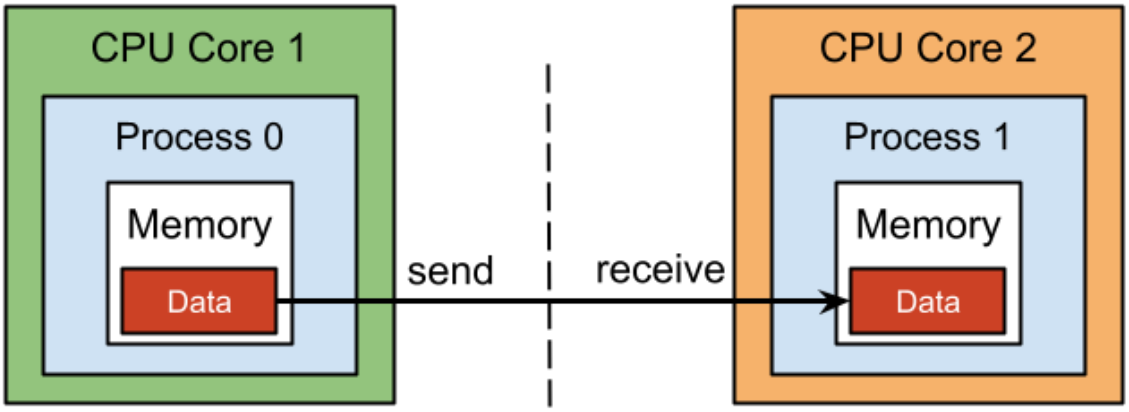
1.3.1 Message Passing Interface (MPI)

Το Message Passing Interface (MPI) είναι ένα τυποποιημένο και φορητό πρότυπο μετάδοσης μηνυμάτων που έχει σχεδιαστεί για να λειτουργεί σε παράλληλες αρχιτεκτονικές υπολογιστών. Το MPI δεν είναι γλώσσα προγραμματισμού. Είναι μια βιβλιοθήκη συναρτήσεων που οι προγραμματιστές μπορούν να καλέσουν από τον κώδικα C, C++ ή Fortran για να γράψουν παράλληλα προγράμματα. Με το MPI, μια συσκευή επικοινωνίας MPI μπορεί να δημιουργηθεί δυναμικά και να έχει πολλαπλές διεργασίες που εκτελούνται

ταυτόχρονα σε ξεχωριστά CPUs. Κάθε διεργασία έχει μια μοναδική κατάταξη MPI (rank) για την αναγνώριση της, το δικό της χώρο μνήμης και εκτελείται ανεξάρτητα από τις άλλες διεργασίες. Οι διεργασίες επικοινωνούν μεταξύ τους περνώντας μηνύματα για την ανταλλαγή δεδομένων. Ο παραλληλισμός εμφανίζεται όταν μια εργασία προγράμματος χωρίζεται σε μικρά κομμάτια και διανέμει αυτά τα κομμάτια μεταξύ των διεργασιών, στις οποίες κάθε διεργασία επεξεργάζεται το μέρος της.

Το MPI παρέχει τρεις διαφορετικές μεθόδους επικοινωνίας που μπορούν να χρησιμοποιήσουν οι διεργασίες MPI για να επικοινωνούν μεταξύ τους. Οι τρόποι επικοινωνίας είναι οι εξής:

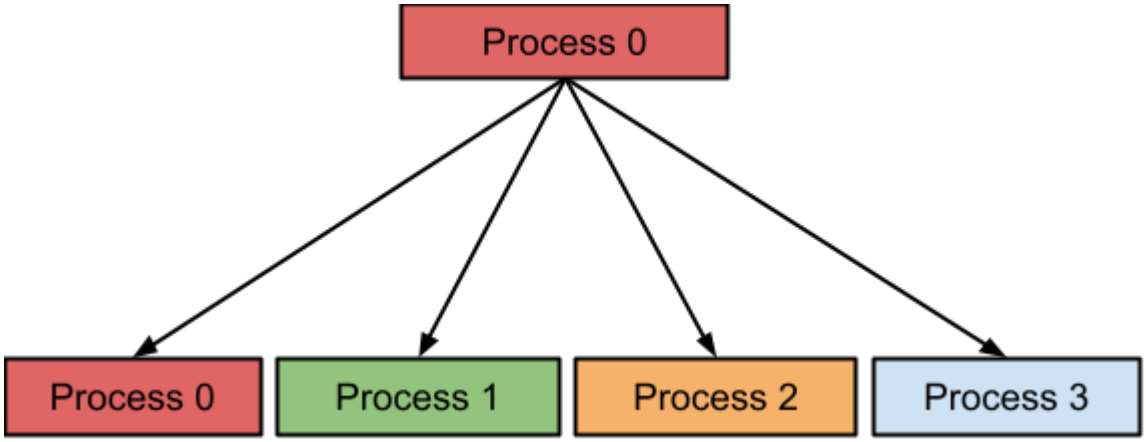
Επικοινωνία από σημείο σε σημείο (Point-to-Point)



Εικόνα 1.1 - Επικοινωνία από σημείο σε σημείο

Η επικοινωνία MPI Point-to-Point είναι η πιο χρησιμοποιούμενη μέθοδος επικοινωνίας στο MPI. Περιλαμβάνει τη μεταφορά ενός μηνύματος από μια διεργασία σε μια συγκεκριμένη διαδικασία στον ίδιο φορέα επικοινωνίας. Το MPI παρέχει αποκλειστική (σύγχρονη) και μη αποκλειστική (ασύγχρονη) επικοινωνία από σημείο σε σημείο. Με την αποκλειστική επικοινωνία, μια διεργασία MPI στέλνει ένα μήνυμα σε μια άλλη διεργασία MPI και περιμένει έως ότου η διαδικασία λήψης λάβει πλήρως και σωστά το μήνυμα πριν συνεχίσει την εργασία της. Από την άλλη πλευρά, μια διαδικασία αποστολής που χρησιμοποιεί επικοινωνία χωρίς αποκλεισμό στέλνει ένα μήνυμα σε μια άλλη διεργασία MPI και συνεχίζει την εργασία της χωρίς να περιμένει για να διασφαλίσει ότι το μήνυμα έχει ληφθεί σωστά από τη διαδικασία λήψης.

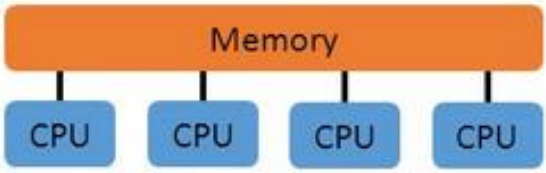
Συλλογική Επικοινωνία



Εικόνα 1.2 - Συλλογική Επικοινωνία

Με αυτόν τον τύπο μεθόδου επικοινωνίας MPI, μια διεργασία εκπέμπει ένα μήνυμα σε όλες τις διεργασίες στον ίδιο φορέα επικοινωνίας, συμπεριλαμβανομένης της ίδιας.

Μονόπλευρη επικοινωνία



Εικόνα 1.3 - Μονόπλευρη επικοινωνία

Με τη μονόπλευρη μέθοδο επικοινωνίας MPI, μια διεργασία μπορεί να έχει απευθείας πρόσβαση στο χώρο μνήμης μιας άλλης διεργασίας χωρίς να την εμπλέκει. Αυτός ο τύπος επικοινωνίας μπορεί συχνά να είναι χρήσιμος για αλγόριθμους στους οποίους ο συγχρονισμός θα ήταν άβολος (π.χ. πολλαπλασιασμός κατανεμημένου πίνακα) ή όπου είναι επιθυμητό οι εργασίες να μπορούν να εξισορροπούν το φορτίο τους ενώ άλλοι επεξεργαστές λειτουργούν με δεδομένα.

Το MPI δεν είναι μια υλοποίηση βιβλιοθήκης από μόνη της, αλλά μάλλον, υπάρχει μια συντριπτική πλειονότητα υλοποιήσεων MPI διαθέσιμες Σε αυτήν την ενότητα, οι πιο ευρέως χρησιμοποιούμενες υλοποιήσεις MPI συζητούνται ως εξής:

Message passing interface chameleon (MPICH)

Το Message passing interface chameleon (MPICH) είναι μια υψηλής απόδοσης, ανοιχτού κώδικα, φορητή εφαρμογή μετάδοσης μηνυμάτων για παράλληλους υπολογισμούς σε συστήματα κατανεμημένης μνήμης. Το MPICH είναι η βάση για ένα ευρύ φάσμα παραγώγων MPI, συμπεριλαμβανομένων των Intel MPI και MVAPICH μεταξύ άλλων. Το MPICH υποστηρίζει διαφορετικά παράλληλα συστήματα από κόμβους πολλαπλών πυρήνων έως συμπλέγματα και μεγάλους υπερυπολογιστές. Υποστηρίζει επίσης δίκτυα υψηλής ταχύτητας και ιδιόκτητα υπολογιστικά συστήματα υψηλής τεχνολογίας.

MPI βιβλιοθήκη της Intel

Η MPI βιβλιοθήκη της Intel εφαρμόζει την προδιαγραφή MPICH. Ένας προγραμματιστής μπορεί να χρησιμοποιήσει τη Βιβλιοθήκη Intel MPI για να δημιουργήσει προηγμένες και πιο σύνθετες παράλληλες εφαρμογές που εκτελούνται σε συμπλέγματα με επεξεργαστές που βασίζονται σε αυτούς της Intel. Επιπλέον, η βιβλιοθήκη Intel MPI παρέχει στους προγραμματιστές τη δυνατότητα να δοκιμάσουν και να συντηρήσουν τις εφαρμογές MPI τους.

MVAPICH

Ανεπτυγμένο από το κρατικό πανεπιστήμιο του Οχάιο, το MVAPICH είναι μια υλοποίηση MPI πάνω από τα πακέτα InfiniBand, Omni-Path, Ethernet iWARP και RoCE. Παρέχει υψηλή απόδοση, επεκτασιμότητα, υποστήριξη ανοχής σφαλμάτων και φορητότητα σε πολλά δίκτυα.

Open Message Passing Interface (OpenMPI)

Το Open Message Passing Interface (OpenMPI) είναι μια εφαρμογή ανοιχτού κώδικα του MPI που διατηρείται από μεγάλες κοινότητες από τη βιομηχανία και τον ακαδημαϊκό κόσμο. Υποστηρίζει πολλές διαφορετικές πλατφόρμες και δίκτυα 32 και 64 bit, συμπεριλαμβανομένων ετερογενών δικτύων. [3]

1.3.2 RabbitMQ

Το RabbitMQ είναι ένα λογισμικό ουράς μηνυμάτων γνωστό και ως μεσίτης μηνυμάτων (message broker) ή διαχειριστής ουρών (queue manager). Είναι ένας διακομιστής όπου ορίζονται ουρές, στον οποίο συνδέονται οι εφαρμογές για να μεταφέρουν ένα μήνυμα ή μηνύματα. Ένα μήνυμα μπορεί να περιλαμβάνει κάθε είδους πληροφορία. Θα μπορούσε, για παράδειγμα, να έχει πληροφορίες σχετικά με μια διαδικασία ή μια εργασία που πρέπει να ξεκινήσει σε μια άλλη εφαρμογή (η οποία θα μπορούσε ακόμη και να είναι σε άλλο

διακομιστή) ή θα μπορούσε να είναι απλώς ένα απλό μήνυμα κειμένου. Το λογισμικό διαχείρισης ουράς αποθηκεύει τα μηνύματα μέχρι να συνδεθεί μια εφαρμογή λήψης και να αφαιρέσει ένα μήνυμα από την ουρά. Στη συνέχεια, η εφαρμογή λήψης επεξεργάζεται το μήνυμα. Αντίθετα με το MPI που ειδικεύεται στην επικοινωνία διαφορετικών εφαρμογών στον ίδιο υπολογιστή, το RabbitMQ καλείται να δράσει ως διαμεσολαβητής στην επικοινωνία μεταξύ εφαρμογών σε διαφορετικούς υπολογιστές με το μοντέλο του publisher-subscriber που υλοποιείται μέσω του AMQP (Advanced Message Queuing Protocol). Αξίζει να σημειωθεί ότι το RabbitMQ έχει την δυνατότητα αποθήκευσης των μηνυμάτων στον σκληρό δίσκο για να διατηρηθεί η ακεραιότητα του συστήματος σε περίπτωση που κάποιος subscriber δεν μπορέσει να διαχειριστεί το μήνυμα για οποιοδήποτε λόγο.

Εικόνα 1.4 – Μοντέλο publisher-subscriber

Τα μηνύματα δεν δημοσιεύονται απευθείας σε μια ουρά. Αντίθετα, ο παραγωγός στέλνει μηνύματα σε μια ανταλλαγή (exchange). Μια ανταλλαγή είναι υπεύθυνη για τη δρομολόγηση των μηνυμάτων σε διαφορετικές ουρές με τη βοήθεια δεσμεύσεων (bindings) και κλειδιών δρομολόγησης (routing keys). Οι δεσμεύσεις είναι σύνδεσμοι μεταξύ μιας ουράς και μιας ανταλλαγής. [4]

Ροή μηνυμάτων στο RabbitMQ

Η ροή μηνυμάτων στο RabbitMQ δουλεύει ως εξής:

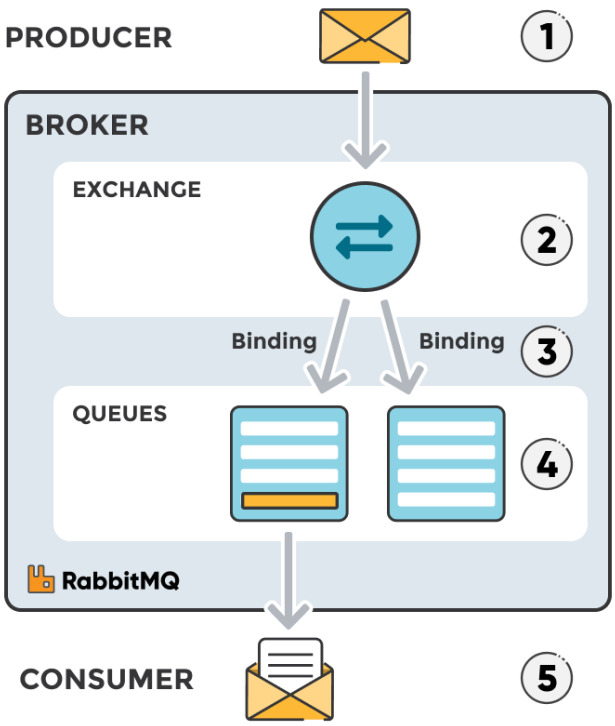
- Ο παραγωγός δημοσιεύει ένα μήνυμα σε μια ανταλλαγή. Κατά τη δημιουργία μιας ανταλλαγής, πρέπει να καθοριστεί ο τύπος της. Αυτό το θέμα θα καλυφθεί αργότερα.

- Η ανταλλαγή λαμβάνει το μήνυμα και είναι πλέον υπεύθυνη για τη δρομολόγηση του μηνύματος. Η ανταλλαγή λαμβάνει υπόψη διαφορετικά χαρακτηριστικά μηνυμάτων, όπως το κλειδί δρομολόγησης, ανάλογα με τον τύπο ανταλλαγής.

- Δημιουργούνται δεσμεύσεις για να κατευθύνουν το μήνυμα από την ανταλλαγή στις αντίστοιχες ουρές.

- Το μήνυμα παραμένει στην ουρά μέχρι να το χειριστεί ένας καταναλωτής (consumer)

- Ο καταναλωτής χειρίζεται το μήνυμα.



Εικόνα 1.5 - Ροή εργασιών RabbitMQ

Τύποι ανταλλαγών

Οι ανταλλαγές χωρίζονται στους παρακάτω 4 τύπους:

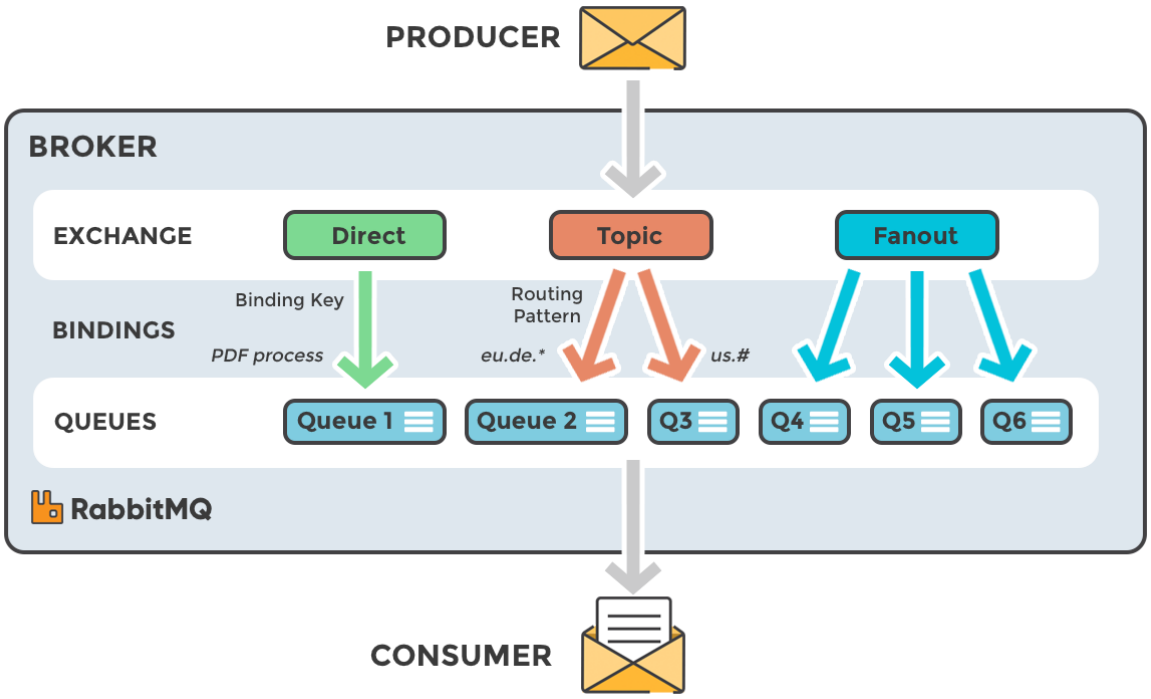
- **Direct:** Το μήνυμα δρομολογείται στις ουρές των οποίων το κλειδί δέσμευσης (binding key) ταιριάζει ακριβώς με το κλειδί δρομολόγησης του μηνύματος.

- **Fanout:** Μια ανταλλαγή fanout δρομολογεί μηνύματα σε όλες τις ουρές που συνδέονται σε αυτό. Το κλειδί δρομολόγησης αγνοείται.

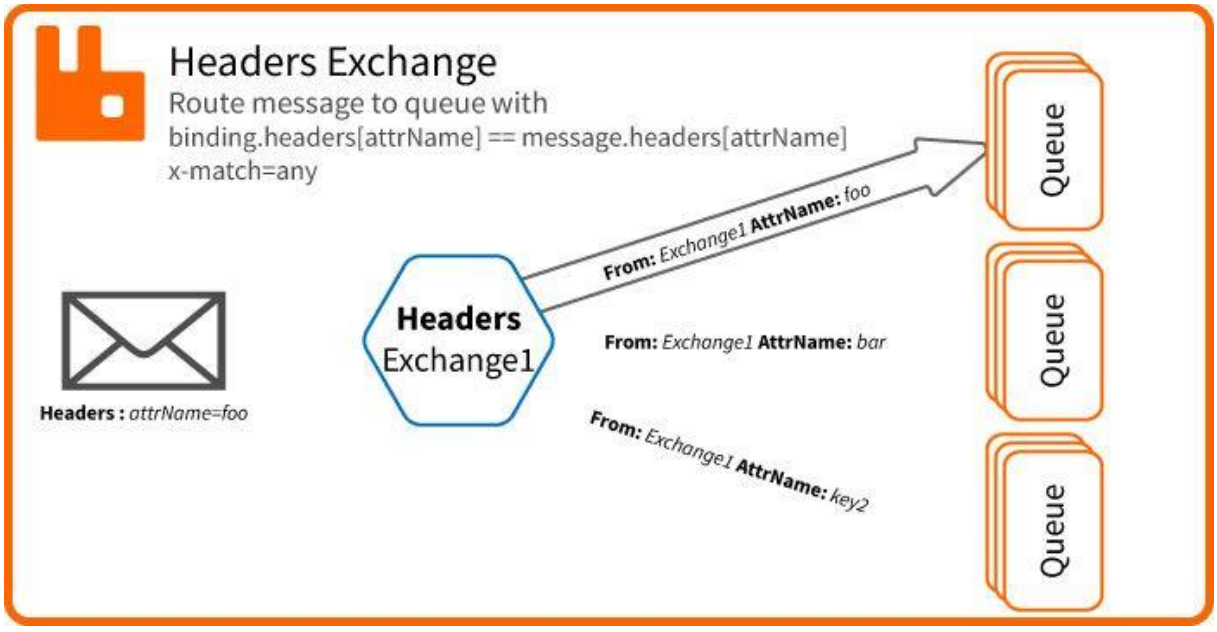
- **Topic:** Η ανταλλαγή topic κάνει μια αντιστοίχιση χαρακτήρων μπαλαντέρ μεταξύ του κλειδιού δρομολόγησης και του μοτίβου δρομολόγησης που καθορίζεται στο binding. Υπάρχουν δύο τύποι μπαλαντέρ που μπορούν να χρησιμοποιηθούν. Η # (δίεση) θα ταιριάξει με μηδέν ή περισσότερες λέξεις και το * (αστέρι) θα ταιριάξει με μία ακριβώς λέξη.

- **Headers:** Η ανταλλαγή headers έχει σχεδιαστεί για δρομολόγηση σε πολλαπλά χαρακτηριστικά που εκφράζονται πιο εύκολα ως κεφαλίδες μηνυμάτων από ότι ένα κλειδί δρομολόγησης. Οι ανταλλαγές κεφαλίδων αγνοούν το χαρακτηριστικό κλειδιού δρομολόγησης. Αντίθετα, τα χαρακτηριστικά που χρησιμοποιούνται για τη δρομολόγηση λαμβάνονται από το χαρακτηριστικό headers του μηνύματος Ένα μήνυμα θεωρείται ότι

ταιριάζει εάν η τιμή της κεφαλίδας ισούται με την τιμή που καθορίζεται κατά τη δέσμευση. [5]



Εικόνα 1.6 - Τύποι ανταλλαγής RabbitMQ



Εικόνα 1.7 - Τύπος ανταλλαγής Headers

1.3.3 Apache Kafka

Τι είναι η ροή συμβάντων (event streaming);

Ένα συμβάν (event) καταγράφει το γεγονός ότι «κάτι συνέβη» στον κόσμο ή σε μια επιχείρηση. Το event streaming λοιπόν είναι το ψηφιακό ισοδύναμο του κεντρικού νευρικού συστήματος του ανθρώπινου σώματος. Από τεχνικής άποψης, το event streaming είναι η πρακτική της λήψης δεδομένων σε πραγματικό χρόνο από πηγές συμβάντων όπως βάσεις δεδομένων, αισθητήρες, κινητές συσκευές, υπηρεσίες cloud και εφαρμογές λογισμικού με τη μορφή ροών συμβάντων. Αποθήκευση αυτών των ροών συμβάντων για μελλοντική ανάκτηση. Χειρισμός, επεξεργασία και αντίδραση στις ροές συμβάντων σε πραγματικό χρόνο καθώς και αναδρομικά· και δρομολόγηση των ροών συμβάντος σε διαφορετικούς προορισμούς, όπως απαιτείται. Το event streaming διασφαλίζει έτσι μια συνεχή ροή και ερμηνεία δεδομένων, έτσι ώστε οι σωστές πληροφορίες να βρίσκονται στο σωστό μέρος, τη σωστή στιγμή.

Σε τι μπορεί χρησιμοποιηθεί η ροή συμβάντων;

Το event streaming εφαρμόζεται σε μια μεγάλη ποικιλία περιπτώσεων χρήσης, σε μια πληθώρα βιομηχανιών και οργανισμών. Οι χρήσεις αυτού περιλαμβάνουν τα εξής:

- Τη διεκπεραίωση πληρωμών και οικονομικών συναλλαγών σε πραγματικό χρόνο, όπως σε χρηματιστήρια, τράπεζες και ασφάλειες.
- Την παρακολούθηση αυτοκινήτων, φορτηγών, στόλων κλπ. Σε πραγματικό χρόνο.
- Για τη συνεχή λήψη και ανάλυση δεδομένων αισθητήρων από συσκευές IoT (internet of things) ή άλλο εξοπλισμό, όπως σε εργοστάσια και αιολικά πάρκα.
- Παρακολούθηση ασθενών σε νοσοκομειακή περίθαλψη και πρόβλεψη αλλαγών στην κατάσταση για να διασφαλιστεί η έγκαιρη θεραπεία σε επείγουσες περιπτώσεις.
- Για σύνδεση, αποθήκευση και διάθεση δεδομένων που παράγονται από διαφορετικά τμήματα μιας εταιρείας.
- Να χρησιμεύσει ως βάση για πλατφόρμες δεδομένων, αρχιτεκτονικές που βασίζονται σε συμβάντα και μικροϋπηρεσίες
- Να συλλέγει και να αντιδρά αμέσως σε αλληλεπιδράσεις και παραγγελίες πελατών, όπως στο λιανικό εμπόριο, στον ξενοδοχειακό και ταξιδιωτικό κλάδο και σε εφαρμογές για κινητά.

Το Apache Kafka ως Πλατφόρμα ροής συμβάντων

Το Kafka συνδυάζει τρεις βασικές δυνατότητες, ώστε να μπορεί κάποιος να εφαρμόσει τις περιπτώσεις χρήσης του για ροή συμβάντων από άκρο σε άκρο με μία μόνο δοκιμασμένη λύση:

1. δημοσίευση (γράψιμο) και εγγραφή (διάβασμα) σε ροές συμβάντων, συμπεριλαμβανομένης της συνεχούς εισαγωγής/εξαγωγής των δεδομένων από άλλα συστήματα.

2. Αποθήκευση ροών συμβάντων με διάρκεια και αξιοπιστία για όσο διάστημα χρειάζεται.

3. Επεξεργασία ροών γεγονότων καθώς συμβαίνουν ή αναδρομικά.

Πως λειτουργεί το Apache Kafka

Το Kafka είναι ένα κατανεμημένο σύστημα που αποτελείται από διακομιστές και πελάτες που επικοινωνούν μέσω ενός πρωτοκόλλου δικτύου TCP υψηλής απόδοσης. Μπορεί να εγκατασταθεί σε υπολογιστές χωρίς λειτουργικό σύστημα, εικονικές μηχανές και κοντέινερ σε περιβάλλοντα εσωτερικής εγκατάστασης καθώς και σε περιβάλλοντα cloud.

Διακομιστές (Servers): Το Kafka εκτελείται ως ένα σύμπλεγμα ενός ή περισσότερων διακομιστών που μπορούν να εκτείνονται σε πολλά κέντρα δεδομένων ή περιοχές cloud. Μερικοί από αυτούς τους διακομιστές σχηματίζουν το στρώμα αποθήκευσης, που ονομάζεται brokers. Άλλοι διακομιστές εκτελούν το Kafka Connect για συνεχή εισαγωγή και εξαγωγή δεδομένων ως ροές συμβάντων για την ενοποίηση του Kafka με τα υπάρχοντα συστήματά, όπως σχεσιακές βάσεις δεδομένων καθώς και άλλα συμπλέγματα Kafka. Για να έχει τη δυνατότητα υλοποίησης κρίσιμων περιπτώσεων χρήσης, ένα σύμπλεγμα Kafka είναι εξαιρετικά επεκτάσιμο και ανεκτικό σε σφάλματα: εάν οποιοσδήποτε από τους διακομιστές του αποτύχει, οι άλλοι διακομιστές θα αναλάβουν την εργασία του για να εξασφαλίσουν συνεχείς λειτουργίες χωρίς απώλεια δεδομένων.

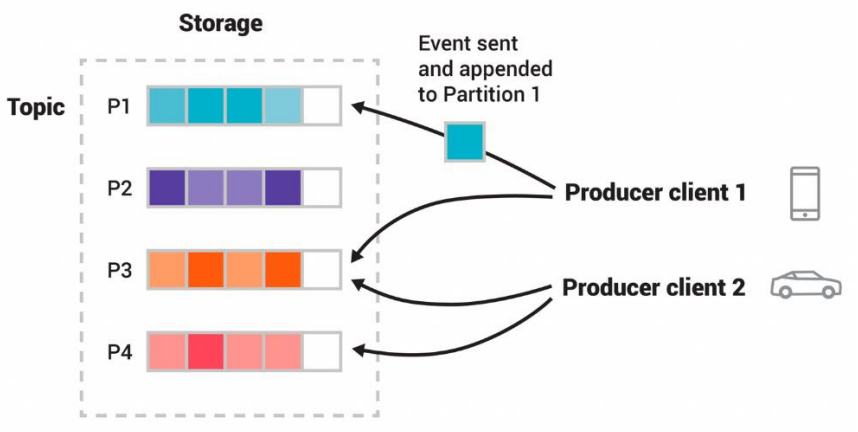
Πελάτες (Clients): Επιτρέπουν στους χρήστες να γράφουν κατανεμημένες εφαρμογές και μικροϋπηρεσίες που διαβάζουν, γράφουν και επεξεργάζονται ροές συμβάντων παράλληλα, σε κλίμακα και με τρόπο ανεκτικό σε σφάλματα, ακόμη και σε περίπτωση προβλημάτων δικτύου ή του υπολογιστή. Οι clients είναι διαθέσιμοι για Java και Scala, συμπεριλαμβανομένης της βιβλιοθήκης Kafka Streams υψηλότερου επιπέδου, για Go, Python, C/C++ και πολλές άλλες γλώσσες προγραμματισμού καθώς και REST API.

Τρόπος λειτουργίας ενός Kafka server

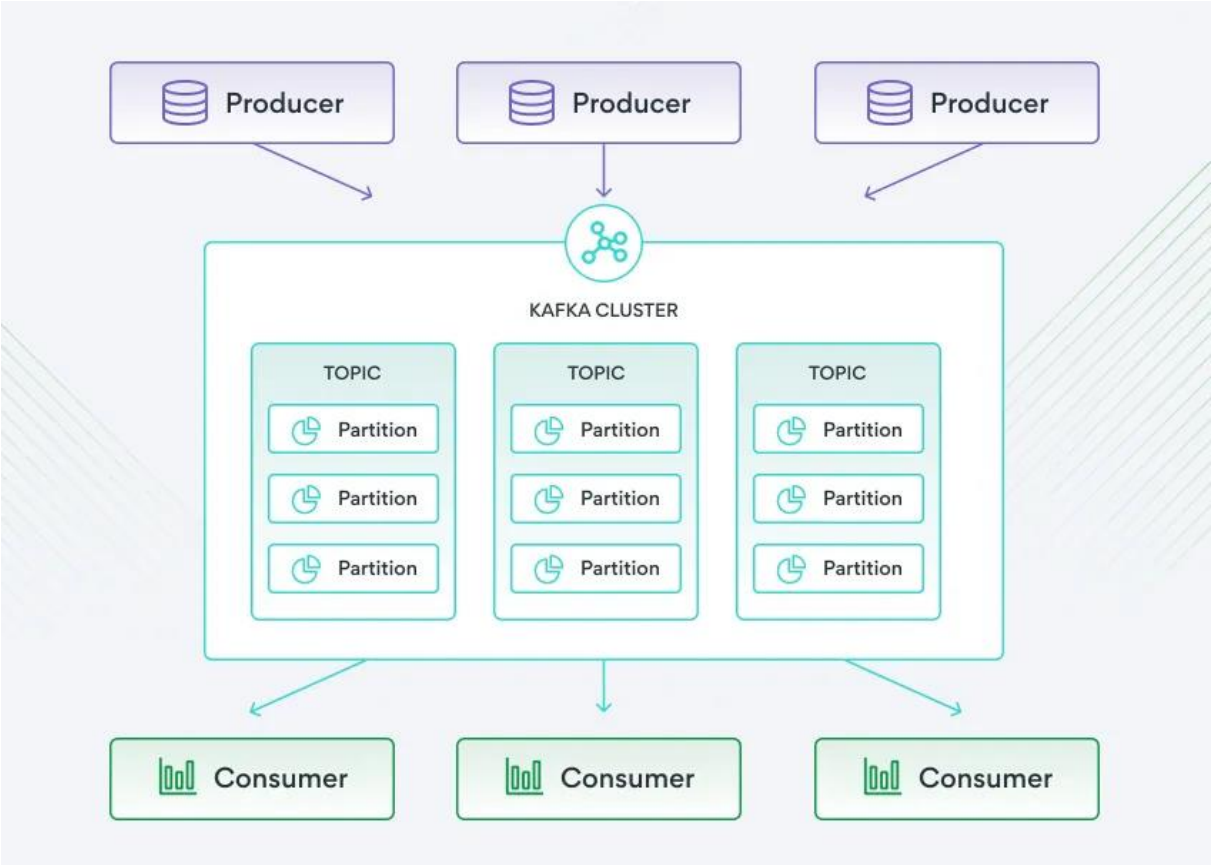
Τα συμβάντα (events) οργανώνονται και αποθηκεύονται σε θέματα (topics). Ένα θέμα είναι παρόμοιο με έναν φάκελο σε ένα σύστημα αρχείων και τα συμβάντα είναι τα αρχεία σε αυτόν τον φάκελο. Τα θέματα στον Kafka είναι πάντα πολλαπλών παραγωγών και πολλών συνδρομητών: ένα θέμα μπορεί να έχει μηδέν, έναν ή πολλούς παραγωγούς που γράφουν συμβάντα σε αυτό, καθώς και μηδέν, έναν ή πολλούς καταναλωτές που εγγράφονται σε αυτά τα συμβάντα. Τα συμβάντα σε ένα θέμα μπορούν να διαβαστούν όσο συχνά χρειάζεται σε αντίθεση με τα παραδοσιακά συστήματα ανταλλαγής μηνυμάτων, τα συμβάντα δεν διαγράφονται μετά την κατανάλωση. Αντίθετα, ο χρήστης ορίζει για πόσο καιρό ο Kafka θα πρέπει να διατηρεί τα συμβάντα μέσω μιας ρύθμισης που μπορεί να είναι διαφορετική ανά θέμα (topic). Όταν περάσει ο χρόνος που έχει οριστεί, τα παλιά συμβάντα διαγράφονται. Η απόδοση του Kafka server είναι ουσιαστικά σταθερή όσον αφορά το μέγεθος των δεδομένων, επομένως η αποθήκευση δεδομένων για μεγάλο χρονικό διάστημα δεν είναι πρόβλημα.

Τα topics είναι χωρισμένα (partitioned), που σημαίνει ότι ένα θέμα απλώνεται σε έναν αριθμό "κουβάδων" που βρίσκονται σε διαφορετικούς μεσίτες Kafka brokers. Αυτή η κατανεμημένη τοποθέτηση των δεδομένων είναι πολύ σημαντική για την επεκτασιμότητα, επειδή επιτρέπει στις εφαρμογές-πελάτες να διαβάζουν και να γράφουν τα δεδομένα από/σε πολλούς brokers ταυτόχρονα. Όταν ένα νέο συμβάν δημοσιεύεται σε ένα θέμα, στην πραγματικότητα προσαρτάται σε ένα από τα partitions του θέματος. Τα συμβάντα με το ίδιο κλειδί συμβάντος (π.χ. αναγνωριστικό πελάτη ή οχήματος) εγγράφονται στο ίδιο partition και ο Kafka εγγυάται ότι οποιοσδήποτε καταναλωτής ενός συγκεκριμένου partition θέματος θα διαβάζει πάντα τα συμβάντα αυτού του partition με την ίδια ακριβώς σειρά που γράφτηκαν.

Για να είναι τα δεδομένα ανεκτικά σε σφάλματα και διαθέσιμα σε κάθε περίπτωση, κάθε topic μπορεί να αναπαραχθεί, έτσι ώστε να υπάρχουν πάντα πολλοί brokers που έχουν αντίγραφο των δεδομένων σε περίπτωση που κάτι πάει στραβά. Μια κοινή ρύθμιση παραγωγής είναι ο συντελεστής αναπαραγωγής 3, δηλαδή, θα υπάρχουν πάντα τρία αντίγραφα των δεδομένων. [6]



Εικόνα 1.8 - Topic χωρισμένο σε 4 partitions



Εικόνα 1.9 - Kafka cluster

Κεφάλαιο 2ο: Μοντέλο της Διπλωματικής

2.1 Αρχιτεκτονική του συστήματος

Στόχος μας είναι η δημιουργία μίας πλατφόρμας επιστημονικών υπολογισμών μεγάλης κλίμακας. Για να είναι δυνατό αυτό το σύστημα θα πρέπει να μπορεί να επεξεργαστεί μεγάλο όγκο δεδομένων σε λογικό χρόνο. Ο τρόπος με τον οποίον θα το καταφέρουμε αυτό είναι με τον διαμοιρασμό των δεδομένων σε πολλούς φυσικούς κόμβους (nodes) οι οποίοι θα τα επεξεργάζονται σε παραλληλία. Εφόσον όμως δεν έχουμε στη διάθεσή μας αρκετούς υπολογιστές για την υλοποίηση ενός τέτοιου δικτύου, για την προσομοίωση του θα γίνει χρήση από 4 VMs (Virtual Machines). Τα προγράμματα φυσικά είναι γραμμένα με τρόπο ούτως ώστε το δίκτυο να είναι επεκτάσιμο σε όσους κόμβους χρειαστεί, χρησιμοποιώντας τον ίδιο ακριβώς κώδικα. Πριν συνεχίσουμε επίσης είναι απαραίτητο να αναφερθούν τα επιμέρους στοιχεία του συστήματος και το ποιες είναι οι αρμοδιότητές τους καθώς και το πώς αλληλοεπιδρούν.

Το βασικό **λογικό** μοντέλο του συστήματος υπολογισμών είναι ένας αφηρημένος N-διάστατος χώρος δεδομένων που ονομάζεται GridSpace. Τέτοιοι χώροι χρησιμοποιούνται ευρέως σε μαθηματικά μοντέλα. Ο χώρος θεωρείται ότι αποτελείται από κελλιά (cells) κατάλληλης τοπολογίας (π.χ. R^n, εξαγωνικά ή άλλά πλέγματα και γράφοι). Ένα υποσύνολο κελλιών (κατάλληλης τοπολογίας, π.χ. παραλληλόγραμμα στον R^2) αποτελούν μια φέτα (slice). Ο συνολικός χώρος (GridSpace) χωρίζεται σε αμοιβεία αποκλειόμενες (mutually exclusive) φέτες. Σε κάθε κελλί μπορούμε να ορίσουμε αυθαίρετο αριθμό μεταβλητών που αντιπροσωπεύουν την κατάσταση του κελλιού. Με αυτή την λογική δομή πολλά (κατανεμημένα) φαινόμενα της επιστήμης και της μηχανικής μπορούν να μελετηθούν.

Υπάρχουν 4 διαφορετικά προγράμματα τα οποία λειτουργούν συγχρόνως, και μοιράζονται τις εργασίες που πρέπει να γίνουν. Τα προγράμματα αυτά είναι τα εξής:

- **Client:** Το client πρόγραμμα είναι υπεύθυνο για τον υπολογισμό, την επεξεργασία της κατάστασης και την εμφάνιση των δεδομένων ενός node, καθώς επίσης και της αρχικοποίησης τους. Έχει επίσης την δυνατότητα να τροποποιήσει ή να διαβάσει οποιοδήποτε κελί των δεδομένων επιλέξει ο χρήστης μέσω συναρτήσεων. Υπάρχουν δύο τρόποι αρχικοποίησης των δεδομένων: μέσω αρχείου όπου τα δεδομένα παίρνουν τις τιμές και τις διαστάσεις του πίνακα που βρίσκεται στο αρχείο, και μέσω τυχαίων τιμών όπου οι τυχαίες τιμές αρχικοποιούνται σε κάθε node ξεχωριστά. Αφού γίνει η αρχικοποίηση και τα προγράμματα των nodes τεθούν σε λειτουργεία, στη συνέχεια ο χρήστης του client προγράμματος έχει τη δυνατότητα να δει τα δεδομένα ενός node μέσω ενός γραφικού περιβάλλοντος που ενημερώνεται με τις αλλαγές των δεδομένων σε πραγματικό χρόνο.

- **Master node:** Οι αρμοδιότητες του master node προγράμματος είναι να μοιράσει τα δεδομένα στα nodes και να δημιουργήσει το partition table. Έπειτα, έχοντας εκτελέσει αυτές τις εργασίες, η δουλειά του γίνεται ίδια με

αυτήν ενός απλού node, η οποία θα εξηγηθεί παρακάτω. Αρχικά, το master node μοιράζει τα δεδομένα σε όσα nodes χρειάζεται έτσι ώστε να είναι επαρκής η μνήμη για την επεξεργασία και αποθήκευσή τους σε κάθε node ξεχωριστά, δίνοντας περισσότερα δεδομένα στα nodes που έχουν μεγαλύτερη επεξεργαστική ισχύ. Αφού γίνει ο διαμοιρασμός των δεδομένων, δημιουργείται το partition table, που είναι ένας πίνακας που δείχνει ποιο κομμάτι των δεδομένων αντιστοιχεί σε ποιο node. Μετά την δημιουργία του στέλνεται στο gateway πρόγραμμα.

- **Slave nodes (daemons):** Τα daemons είναι nodes που έχουν το ίδιο πρόγραμμα να τρέχει. Υπολογίζουν την επόμενη κατάσταση των δεδομένων με βάση το μαθηματικό μοντέλο της εφαρμογής. Για να το κάνουν αυτό παίρνουν μηνύματα από τα γειτονικά nodes στο grid των δεδομένων και τα επεξεργάζονται. Μόλις υπολογιστεί η επόμενη κατάσταση των δεδομένων, στέλνεται με τη σειρά της στα γειτονικά nodes. Αυτές οι εργασίες επαναλαμβάνονται μέχρι να διακοπεί η εκτέλεση του προγράμματος.

- **Gateway:** Το gateway καλείται να δράσει ως διαμεσολαβητής μεταξύ του client προγράμματος και των nodes. Είναι το πρόγραμμα που λαμβάνει τις μεταβλητές για την αρχικοποίηση του λογικού χώρου από τον client και τις αποστέλλει στο Master node, το οποίο θα τις διαχειριστεί περαιτέρω. Επίσης χρησιμοποιεί το partition table για να δρομολογήσει στο σωστό node την αίτηση για διάβασμα ή τροποποίηση συγκεκριμένου κελιού, η οποία προέρχεται από τον client. Κυριώτερα όμως, στέλνει στον client τα δεδομένα που λαμβάνει από το node που αυτός έχει επιλέξει να παρακολουθήσει για κάθε λογικό βήμα, μέχρις ότου ο client να αποφασίσει να σταματήσει να το παρακολουθεί.

Για την αρχικοποίηση του κόσμου καθώς επίσης και για το διάβασμα ή την τροποποίηση ενός κελιού έχουν δημιουργηθεί ανάλογες συναρτήσεις, οι οποίες λειτουργούν ως API που χρησιμοποιεί ο client για την περάτωση αυτών των ενεργειών. Οι συναρτήσεις αυτές είναι οι εξής:

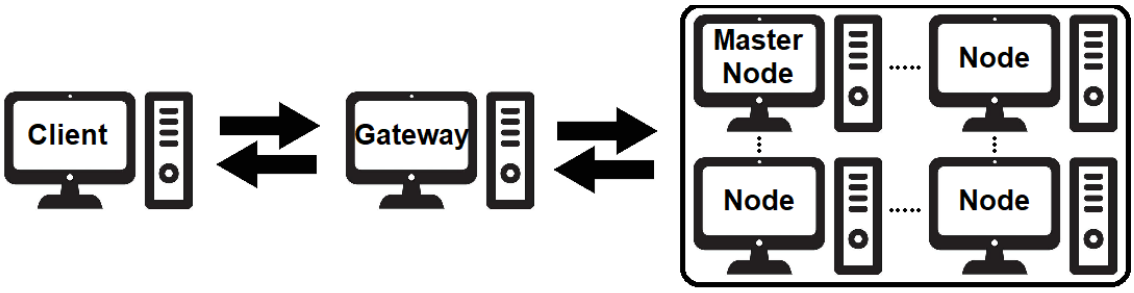
- **initWorld:** Η συνάρτηση για την δημιουργία του λογικού χώρου. Δίνει την επιλογή της δημιουργίας ενός χώρου με τυχαίες τιμές ή με προκαθορισμένες τιμές μέσω αρχείου. Δίνει επίσης τη δυνατότητα επιλογής του αριθμού των nodes στα οποία θα διαμοιραστούν τα δεδομένα ή του δυναμικού διαμοιρασμού με βάση την μνήμη και επεξεργαστική ισχύ των nodes.

- **readCell:** Αυτή η συνάρτηση θα λάβει ως παράμετρο την απόλυτη θέση του κελιού που επιθυμεί ο client να διαβάσει και έπειτα θα στείλει στον Gateway αίτημα για διάβασμα. Ο Gateway με τη σειρά του θα στείλει το αίτημα στο κατάλληλο node με την αντιστοιχη σχετική θέση του κελιου (πληροφορίες που βρίσκει μέσω του partition table) και το node επιστρέφει τελικά στον client την τιμή του κελιού.

- **writeCell:** Μέσω αυτής της συνάρτησης γίνεται η τροποποίηση ενός κελιού. Ο τρόπος λειτουργίας της είναι ίδιος με αυτήν της readCell με την διαφορά ότι

τροποποιεί κελί ενός node αντί να το διαβάζει και δεν αποστέλλει απάντηση πίσω στον client.

Κάθε node πρόγραμμα συμπεριλαμβανομένου και του master node τρέχει σε διαφορετικό VM. Τα VM αυτά χρησιμοποιούν λειτουργικό σύστημα Ubuntu. Το gateway και client πρόγραμμα αντιθέτως τρέχουν στο λογισμικό του κεντρικού υπολογιστή, το οποίο είναι το Windows 10. Με αυτόν τον τρόπο δοκιμάζεται η δυνατότητα της προγραμματιστικής γλώσσας Nim να δουλέψει σε διαφορετικές πλατφόρμες και λογισμικά χωρίς τροποποιήσεις στον κώδικα. Η Nim χρησιμοποιήθηκε γιατί είναι μια γλώσσα που παράγει κώδικα C υψηλής ταχύτητας και είναι multi-platform χωρίς την ανάγκη για Java Vistual Machines ή παρόμοιες τεχνολογίες. Επιπλέον είναι πιο κοντά στο μοντέλο του Object Oriented Programming από ότι η C. Το τελικό αποτέλεσμα είναι ότι παράγεται κώδικας C ως ενδιάμεση αναπαράσταση ο οποίος μεταγλωτίζεται βέλτιστα από τον C compiler του κάθε κόμβου.



Εικόνα 2.1 - Τοπολογία του δικτύου

2.2 Αλγόριθμος μέτρησης υπολογιστικής ισχύος

Για να έχει το master node την δυνατότητα να μοιράσει τα δεδομένα με βάση την μνήμη και την επεξεργαστική ισχύ, πρέπει να συλλέξει αυτές τις πληροφορίες από κάθε node συμπεριλαμβανομένου και του εαυτού του. Το να βρούμε την ποσότητα της ελεύθερης μνήμης RAM ενός κόμβου είναι μία απλή εργασία και γίνεται εύκολα με το κάλεσμα μίας συνάρτησης που είναι διαθέσιμη μέσω μίας βιβλιοθήκης. Η εύρεση της επεξεργαστικής ισχύς όμως είναι πιο περίπλοκη εργασία. Για να την φέρουμε σε πέρας έγινε η υλοποίηση του αλγορίθμου μέτρησης υπολογιστικής ισχύος. Ο αλγόριθμος αυτός συλλέγει τα δεδομένα που χρειάζεται με δύο τρόπους: με αξιολόγηση του επεξεργαστή (CPU) του κόμβου βάσει ενός πίνακα δεδομένων που περιέχει βαθμολογίες για κάθε επεξεργαστή, και με δυναμική μέτρηση της ταχύτητας υπολογισμού του κόμβου. Αφού γίνει η συλλογή αυτών των δεδομένων, μετατρέπονται σε μία τιμή που είναι στην ουσία ο μέσος όρος των δύο αποτελεσμάτων.

Ο πίνακας αξιολογήσεων των επεξεργαστών είναι διαθέσιμος μέσω της σελίδας https://www.cpubenchmark.net/cpu_list.php και είναι μια λίστα που για κάθε επεξεργαστή υπάρχει το όνομα, η βαθμολογία του, η κατάταξη του σε σχέση με τους υπόλοιπους, η τιμή του, και η αξία του, η οποία έχει να κάνει με την απόδοση του σε συνάρτηση με την τιμή του. Για τις απαιτήσεις του αλγορίθμου θα χρειαστούμε μόνο την βαθμολογία και το όνομα του επεξεργαστή.

Για την δυναμική μέτρηση της ταχύτητας υπολογισμού κάθε κόμβου θα γίνει χρήση ενός αλγορίθμου που ονομάζεται *Κόσκινο του Ερατοσθένη*. Στα μαθηματικά, το Κόσκινο του Ερατοσθένη είναι ένας απλός αλγόριθμος για την εύρεση όλων των πρώτων αριθμών μέχρι έναν συγκεκριμένο ακέραιο. Η εύρεση όλων των πρώτων αριθμών που είναι μικρότεροι ή ίσοι από έναν ακέραιο *n*, σύμφωνα με τη μέθοδο του Ερατοσθένη, γίνεται ως εξής:

1. Δημιουργούμε μια λίστα από διαδοχικούς ακέραιους από το 2 μέχρι το *n*: (2, 3, 4, ..., *n*).

2. Αρχικά, έστω ότι το *p* είναι ίσο με 2, τον 1ο πρώτο αριθμό.

3. Διαγράφουμε από τη λίστα όλα τα πολλαπλάσια του *p* που είναι μικρότερα ή ίσα με *n*. (*2p, 3p, 4p, κτλ*)

4. Βρίσκουμε τον 1ο αριθμό που απομένει στη λίστα μετά τον *p* (αυτός ο αριθμός είναι ο επόμενος πρώτος αριθμός) και αντικαθιστούμε το *p* με αυτόν τον αριθμό.

5. Επαναλαμβάνουμε τα βήματα 3 και 4 μέχρι το p^2 να είναι μεγαλύτερο από *n*.

6. Όλοι οι αριθμοί που απομένουν στη λίστα είναι πρώτοι αριθμοί.

Όταν το Κόσκινο του Ερατοσθένη έχει βγάλει το αποτέλεσμα του, καταγράφεται ο χρόνος που χρειάστηκε για τον υπολογισμό του. Αυτός ο χρόνος με τη σειρά του μετατρέπεται σε έναν αριθμό παρόμοιο με αυτόν της βαθμολογίας των επεξεργαστών. Ο χρόνος υπολογισμού και η βαθμολογία που παράγεται πρέπει δηλαδή να είναι αντιστρόφως ανάλογες τιμές.

Τελικά, ο αλγόριθμος μέτρησης υπολογιστικής ισχύος λειτουργεί με τα εξής βήματα:

1. Κάθε κόμβος υπολογίζει δυναμικά την επεξεργαστική του ισχύ με τη βοήθεια του κόσκινου του Ερατοσθένη.
2. Οι κόμβοι daemons στέλνουν στον master κόμβο το όνομα του επεξεργαστή τους, και το αποτέλεσμα από τον δυναμικό υπολογισμό, καθώς επίσης και την ποσότητα της ελεύθερης μνήμης τους.
3. Ο master κόμβος αναθέτει σε κάθε όνομα επεξεργαστή των κόμβων (συμπεριλαμβανομένου και του δικού του) την βαθμολογία που του αντιστοιχεί μεσώ του πίνακα αξιολογήσεων και έπειτα αυτή η βαθμολογία συμψηφίζεται με την επεξεργαστική ισχύ που υπολογίστηκε δυναμικά για να βρεθεί ο τελικός βαθμός.

Κεφάλαιο 3ο: Εγκατάσταση και Υλοποίηση

3.1 Εικονικές μηχανές (Virtual machines)

Το Virtual Machine (VM) είναι ένας υπολογιστικός πόρος που χρησιμοποιεί λογισμικό αντί για φυσικό υπολογιστή για την εκτέλεση προγραμμάτων και την ανάπτυξη εφαρμογών. Μία ή περισσότερες εικονικές μηχανές "επισκέπτες" μπορούν να λειτουργούν σε μια φυσική μηχανή "κεντρικού υπολογιστή" και να εκτελούν εργασίες. Κάθε εικονική μηχανή εκτελεί το δικό της λειτουργικό σύστημα και λειτουργεί ξεχωριστά από τις άλλες, ακόμα και όταν όλες εκτελούνται στον ίδιο κεντρικό υπολογιστή. Αυτό σημαίνει ότι, για παράδειγμα, μια εικονική μηχανή MacOS μπορεί να εκτελεστεί σε φυσικό υπολογιστή. Η εικονική μηχανή εκτελείται ως διαδικασία (process) σε ένα παράθυρο εφαρμογής, παρόμοια με οποιαδήποτε άλλη εφαρμογή, στο λειτουργικό σύστημα της φυσικής μηχανής. Τα βασικά αρχεία που συνθέτουν μια εικονική μηχανή περιλαμβάνουν ένα αρχείο καταγραφής, ένα αρχείο ρυθμίσεων NVRAM, ένα αρχείο εικονικού δίσκου και ένα αρχείο διαμόρφωσης (configuration file). [7]

Για τις ανάγκες τις εφαρμογής θα δημιουργήσουμε 4 όμοιες εικονικές μηχανές. Το λειτουργικό σύστημα που θα τρέχουν είναι το Ubuntu 14. Η εφαρμογή που θα χρησιμοποιήσουμε για να εκτελέσουμε και να διαχειριστούμε αυτές τις εικονικές μηχανές είναι το Oracle VirtualBox. Το Oracle VirtualBox είναι λογισμικό εικονικοποίησης ανοιχτού κώδικα που επιτρέπει στους χρήστες να εκτελούν πολλαπλά λειτουργικά συστήματα (εικονικές μηχανές) σε μία μόνο συσκευή (φυσική μηχανή). Το λογισμικό αυτό είναι διαθέσιμο για λήψη και εγκατάσταση από την αντίστοιχη ιστοσελίδα της Oracle.

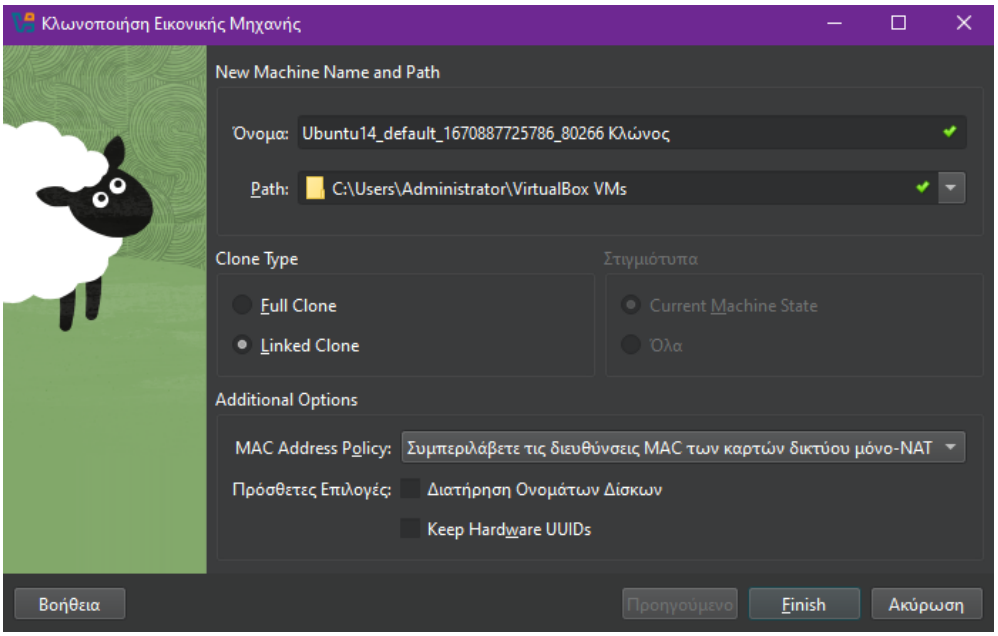
Για την εύρεση της κατάλληλης “εικόνας” λογισμικού που χρειαζόμαστε (Ubuntu 14), και την εγκατάσταση αυτού ως νέα εικονική μηχανή θα γίνει χρήση ενός λογισμικού που λέγεται Vagrant. Το Vagrant είναι ένα πρόγραμμα που χρησιμοποιείται για τη δημιουργία και τη συντήρηση φορητών περιβαλλόντων ανάπτυξης εικονικού λογισμικού. Επιτρέπει την διαχείριση και δημιουργία νέων εικονικών μηχανών μέσω της γραμμής εντολών, και τις ρυθμίσεις πολλαπλών εικονικών μηχανών μέσω του ίδιου αρχείου διαμόρφωσης. Για την εγκατάσταση της εικόνας λοιπόν οι εξής εντολές πρέπει να εκτελεστούν στον φάκελο που θέλουμε να αποθηκευτεί η εικονική μηχανή, μέσω της γραμμής εντολών:

```
vagrant box add ubuntu/trusty64
vagrant init ubuntu/trusty64
vagrant plugin install vagrant-vbguest
vagrant up
```

Έχοντας εκτελέσει αυτά τα βήματα, η εικονική μηχανή που δημιουργήθηκε θα εμφανιστεί στο γραφικό περιβάλλον του VirtualBox. Για την πιο ομαλή λειτουργιά της εικονικής μηχανής όμως, πριν από οποιαδήποτε άλλη εργασία, ενδείκνυται να εκτελεστούν κάποιες εντολές που σκοπό έχουν την ενημέρωση των πακέτων του εικονικού λειτουργικού συστήματος στην νεότερη έκδοση, και την εγκατάσταση λογισμικών που είναι απαραίτητα για την ανάπτυξη εφαρμογών (πχ μεταγλωττιστές):

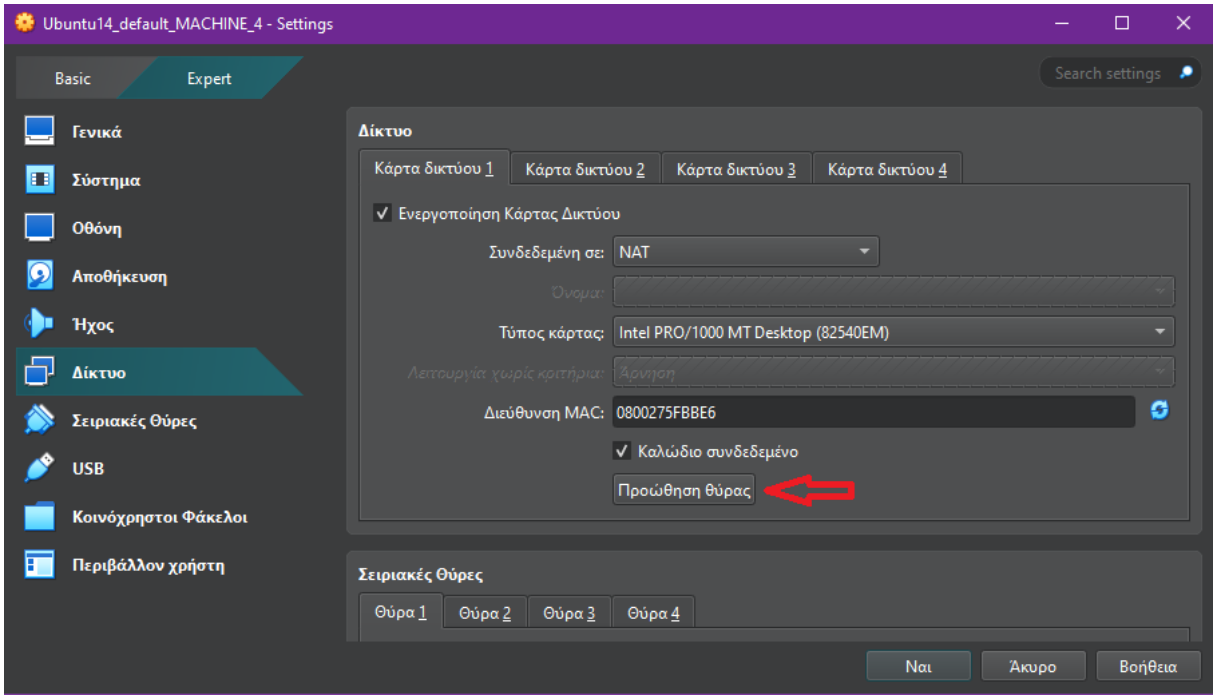
```
sudo apt-get update
sudo apt update
sudo apt upgrade
sudo apt install build-essential
sudo apt-get install manpages-dev
sudo apt install git
```

Έτσι, η εικονική μηχανή είναι πλέον έτοιμη για χρήση. Για να δημιουργήσουμε τις υπόλοιπες 3, θα χρησιμοποιήσουμε το γραφικό περιβάλλον του VirtualBox, το οποίο μας δίνει την δυνατότητα της κλωνοποίησης. Μέσω της κλωνοποίησης μπορούμε να δημιουργήσουμε ένα ακριβές αντίγραφο μίας εικονικής μηχανής που θα λειτουργεί ανεξάρτητα από την αρχική. Πιο συγκεκριμένα, θα δημιουργήσουμε έναν συνδεδεμένο κλώνο της ήδη υπάρχουσας εικονικής μηχανής. Οι συνδεδεμένοι κλώνοι έχουν την ιδιότητα να χρησιμοποιούν το ίδιο αρχείο εικονικού δίσκου με την εικονική μηχανή που κλωνοποιείται. Με αυτόν τον τρόπο, εξασφαλίζεται η ακεραιότητα και συνοχή του συστήματος, εφόσον όλες οι εικονικές μηχανές θα τρέχουν στην ουσία το ίδιο ακριβώς λειτουργικό παρόλο που είναι ξεχωριστές οντότητες.

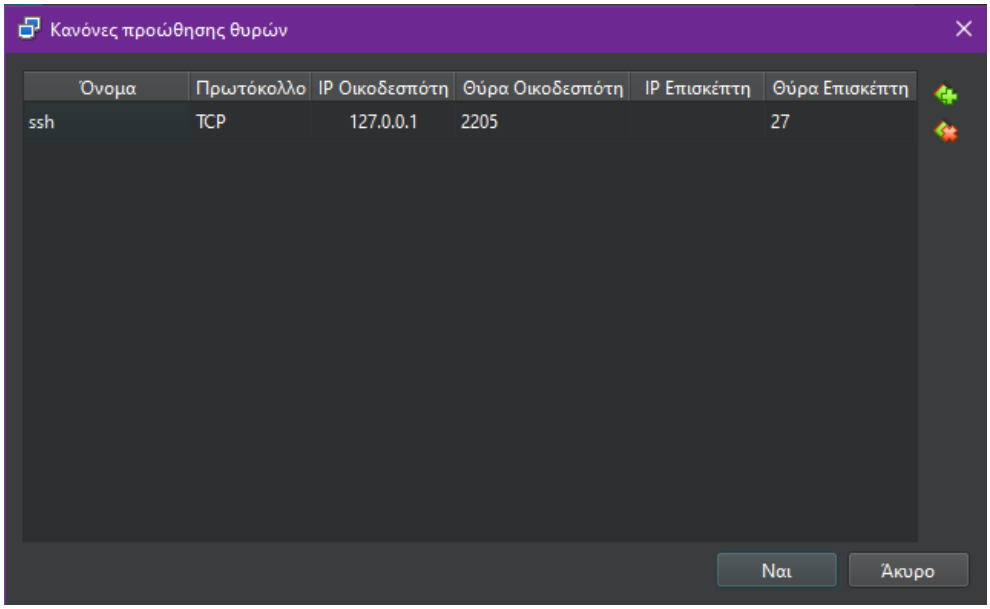


Εικόνα 3.1 – Διαδικασία Κλωνοποίησης εικονικής μηχανής

Για να λειτουργήσουν συγχρόνως 2 ή περισσότερες εικονικές μηχανές πρέπει να έχουν όλες διαφορετική προώθηση θύρας (port forwarding). Ο σκοπός της προώθησης θύρας είναι να παρέχει στην εικονική μηχανή πρόσβαση στο διαδίκτυο μέσω της σύνδεσης της μηχανής-οικοδεσπότη. Η τροποποίηση του port forwarding γίνεται από το αντίστοιχο παράθυρο ρυθμίσεων δικτύου της εικονικής μηχανής.

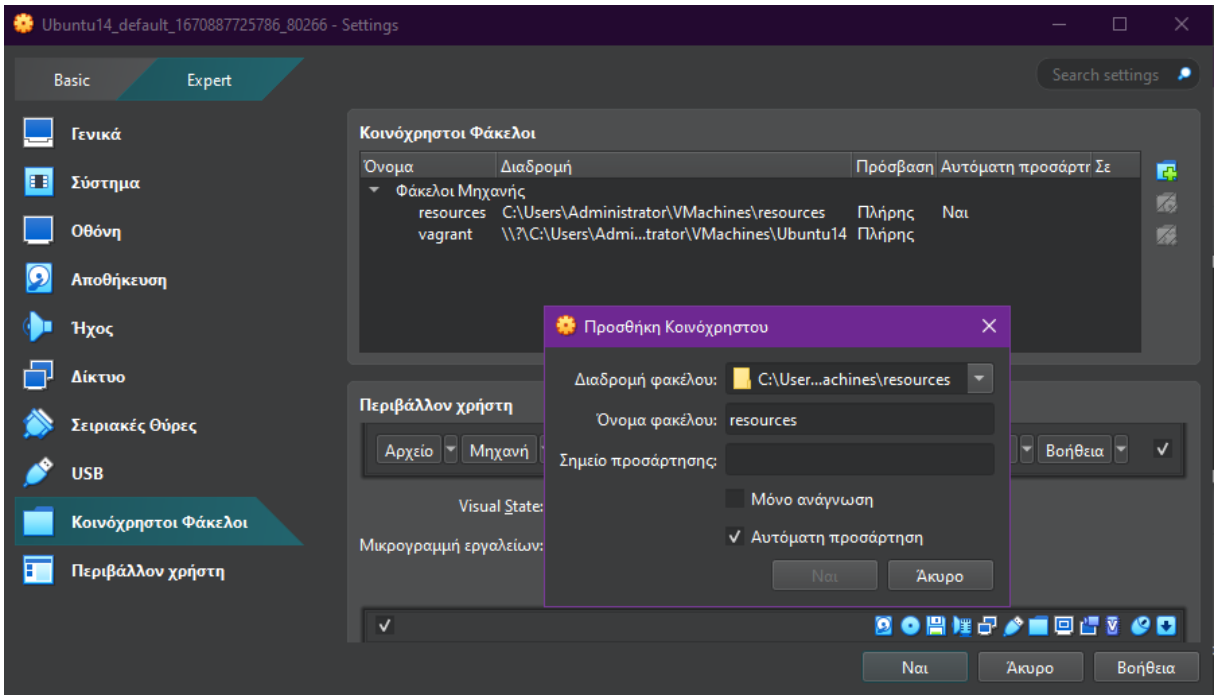


Εικόνα 3.2 - Ρυθμίσεις δικτύου



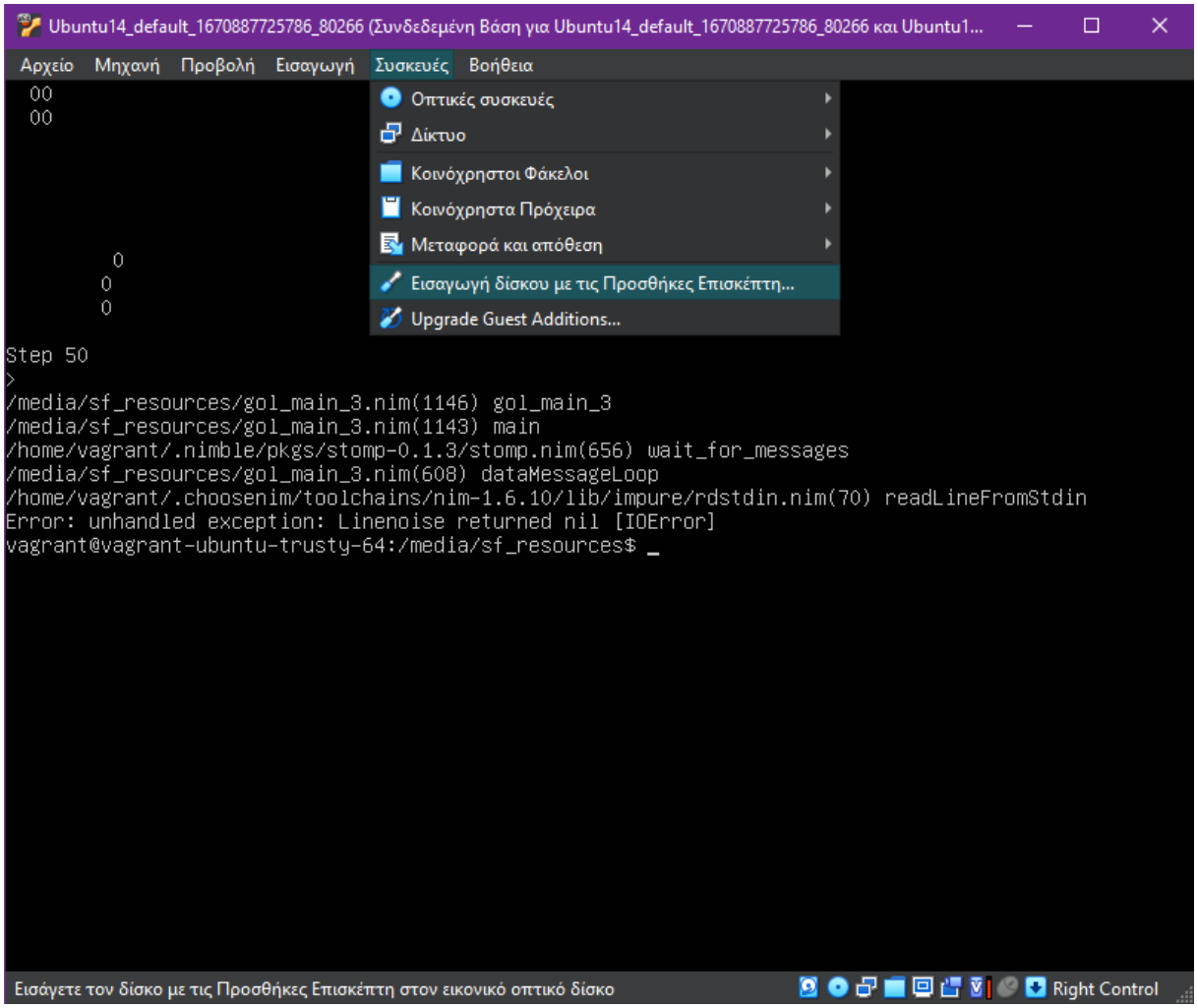
Εικόνα 3.3 - Ρυθμίσεις προώθησης θύρας

Το τελευταίο βήμα για την ολοκλήρωση της ρύθμισης των εικονικών μηχανών είναι να θέσουμε για την κάθε μία τον κοινόχρηστο φάκελο. Με τους κοινόχρηστους φακέλους, είναι δυνατή η πρόσβαση στα αρχεία του κεντρικού υπολογιστή από το σύστημα της εικονικής μηχανής. Οι κοινόχρηστοι φάκελοι βρίσκονται φυσικά στον κεντρικό υπολογιστή και στη συνέχεια μοιράζονται με τον επισκέπτη, ο οποίος χρησιμοποιεί ένα ειδικό πρόγραμμα οδήγησης συστήματος αρχείων για να επικοινωνήσει με τον κεντρικό υπολογιστή. Στο παράθυρο διαλόγου για την προσθήκη κοινόχρηστου φακέλου, η αυτόματη προσάρτηση πρέπει να είναι επιλεγμένη.



Εικόνα 3.4 - Προσάρτηση κοινόχρηστου φακέλου

Έχοντας προσαρτήσει τον φάκελο, η διαδικασία εγκατάστασης του οδηγού προγράμματος στην εικονική μηχανή γίνεται απλά από το μενού του παραθύρου της μετά την εκκίνηση.



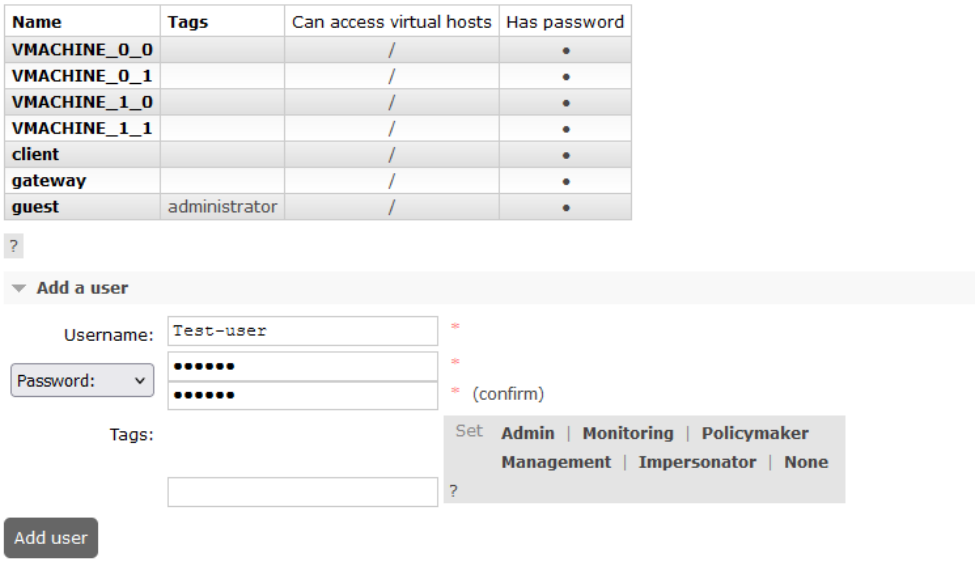
Εικόνα 3.5 - Εγκατάσταση Οδηγού προγράμματος

Στα VM αυτά που έχουν πλέον εγκατασταθεί και ρυθμιστεί, θα γίνει ο διαμοιρασμός των δεδομένων στα οποία θα εκτελεί εργασίες το πρόγραμμα. Ο χώρος των δεδομένων ονομάζεται GridSpace (από εδώ και πέρα θα ονομάζουμε το 2D GridSpace ως GridWorld) και είναι ένα δισδιάστατο πλέγμα στο οποίο κάθε μονάδα του πλέγματος περιέχει μία μεταβλητή ή ένα σύνολο μεταβλητών στις οποίες αποθηκεύονται τα δεδομένα τις εφαρμογής για τους επιστημονικούς υπολογισμούς που πρέπει να διαχειριστεί. Στην πιο απλή μορφή του, που είναι αυτή που θα εξετάσουμε, κάθε μονάδα αποθηκεύει μία μόνο μεταβλητή που παίρνει τιμές 0 ή 1 (true ή false). Μετά τη δημιουργία του, ο λογικός χώρος GridWorld χωρίζεται σε slices που το κάθε ένα ανατίθεται σε ξεχωριστό VM. Τα slices είναι κυριολεκτικά κομμάτια του αρχικού λογικού χώρου και χωρίζονται με τρόπο ώστε να παίρνουν περισσότερες σειρές του δισδιάστατου πίνακα από τον οποίο αποτελείται το GridWorld, τα VM (nodes) που έχουν περισσότερη υπολογιστική ισχύ από τα υπόλοιπα, υπό την προϋπόθεση ότι έχουν αρκετή μνήμη για να τις υποστηρίξουν.

3.2 Εγκατάσταση και ρύθμιση του RabbitMQ

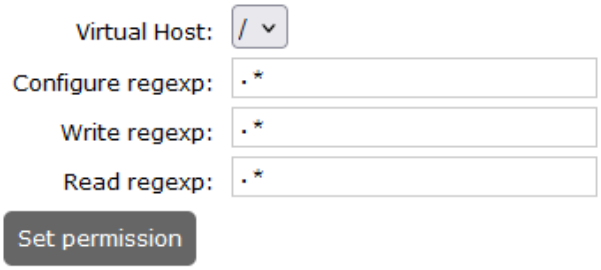
Το RabbitMQ είναι το λογισμικό που θα χρησιμοποιηθεί για την αποστολή μηνυμάτων μεταξύ των διαφορετικών εικονικών μηχανών και του κεντρικού υπολογιστή. Το πρόγραμμα αυτό θα εγκατασταθεί στον κεντρικό υπολογιστή και θα δρα ως διακομιστής στον οποίον θα συνδέονται οι εφαρμογές που θέλουν να εξυπηρετηθούν. Η εγκατάστασή του είναι δυνατή μέσω της σελίδας GitHub του RabbitMQ. Για να λειτουργήσει το RabbitMQ επίσης χρειάζεται να υπάρχει εγκατεστημένη στο σύστημα η γλώσσα προγραμματισμού Erlang.

Μόλις εγκατασταθεί, το RabbitMQ δημιουργεί έναν χρήστη guest με κωδικό πρόσβασης guest. Από προεπιλογή, αυτά τα διαπιστευτήρια μπορούν να χρησιμοποιηθούν μόνο όταν συνδέεται κάποιος στον διακομιστή ως localhost. Είναι αναγκαία λοιπόν, για την προσομοίωση ενός δικτύου υπολογιστών, η δημιουργία ενός χρήστη για κάθε εφαρμογή (υπολογιστή) που θα συνδεθεί σε αυτό. Η δημιουργία αυτών των χρηστών γίνεται απλά μέσω του πίνακα διαχειριστή της σελίδας localhost:15672.



Εικόνα 3.6 - Δημιουργία χρηστών RabbitMQ

Τώρα που δημιουργήθηκαν οι χρήστες για κάθε εφαρμογή, για να μπορέσουν να έχουν πρόσβαση στις λειτουργίες του RabbitMQ που χρειαζόμαστε, πρέπει να τους δοθούν τα απαραίτητα δικαιώματα. Η σελίδα για την επεξεργασία δικαιωμάτων ενός χρήστη είναι διαθέσιμη επιλέγοντας το όνομα του χρήστη από την λίστα που βρίσκεται στον πίνακα διαχειριστή.

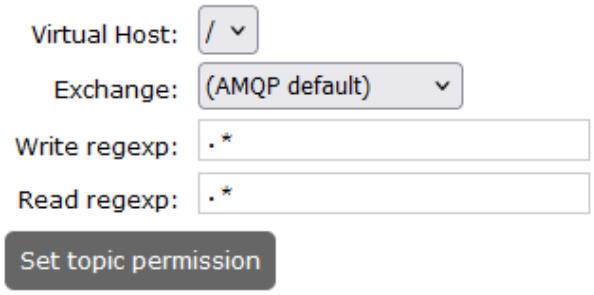


Εικόνα 3.7 - Δικαιώματα χρήστη

Το πρωτόκολλο που θα χρησιμοποιήσουμε για την αποστολή μηνυμάτων είναι το STOMP (Simple Text Orientated Messaging Protocol.). Το STOMP παρέχει μια διαλειτουργική μορφή καλωδίου έτσι ώστε οι πελάτες STOMP να μπορούν να επικοινωνούν με οποιονδήποτε μεσίτη μηνυμάτων STOMP για να παρέχουν εύκολη και ευρέως διαδεδομένη διαλειτουργικότητα ανταλλαγής μηνυμάτων μεταξύ πολλών γλωσσών και πλατφορμών. Όπως αναφέρουν και τα αρχικά του, το πρωτόκολλο STOMP είναι φτιαγμένο έτσι ώστε να μην προσφέρει εξειδικευμένες λειτουργίες, αλλά να κάνει αντί αυτού την αποστολή μηνυμάτων απλούστερη εργασία λόγω του γεγονότος ότι δέχεται για περιεχόμενο μηνύματος απλό κείμενο. Το λογισμικό RabbitMQ διαθέτει ένα πρόσθετο (plugin) που του δίνει τη δυνατότητα να δράσει ως μεσίτης μηνυμάτων κάνοντας χρήση του πρωτοκόλλου STOMP. Η εγκατάστασή του γίνεται με την εξής εντολή στην γραμμή εντολών:

```
rabbitmq-plugins enable rabbitmq_stomp
```

Μετά την εγκατάστασή του προσθέτου, η επιλογή προσθήκης των αντίστοιχων δικαιωμάτων εμφανίζεται στον πίνακα διαχειριστή και πρέπει να δοθούν για κάθε χρήστη



Εικόνα 3.8 - Δικαιώματα χρήστη STOMP plugin

Το RabbitMQ και οι επιμέρους χρήστες είναι πλέον πλήρως ρυθμισμένοι. Τα μηνύματα κάθε χρήστη θα στέλνονται από προεπιλογή στην ανταλλαγή τύπου topic και θα δρομολογούνται ανάλογα, με τον τρόπο που εξηγήθηκε στο κεφάλαιο 1.

3.3 Η γλώσσα Nim

Η γλώσσα που θα χρησιμοποιήσουμε για την δημιουργία των εφαρμογών είναι η NIM. Η Nim είναι μια γενικής χρήσης, στατικά ορισμένων τύπων , μεταγλωττισμένη γλώσσα προγραμματισμού συστήματος υψηλού επιπέδου. Η Nim υποστηρίζει μεταπρογραμματισμό, λειτουργικό προγραμματισμό, μετάδοση μηνυμάτων, διαδικαστικό προγραμματισμό και αντικειμενοστραφή προγραμματισμό, παρέχοντας πολλές δυνατότητες όπως δημιουργία κώδικα χρόνου μεταγλώττισης, αλγεβρικούς τύπους δεδομένων, διεπαφή ξένης συνάρτησης (FFI) με C, C++, Objective-C, και JavaScript, και υποστήριξη μεταγλώττισης σε αυτές τις γλώσσες ως ενδιάμεσες αναπαραστάσεις. Ένα πρόγραμμα Nim τρέχει το ίδιο γρήγορα με ένα όμοιο πρόγραμμα C και έχει τη δυνατότητα να μεταγλωττιστεί και να εκτελεστεί σε διαφορετικά λειτουργικά συστήματα. Η γλώσσα Nim απαιτεί να υπάρχει εγκατεστημένος στο σύστημα ένας μεταγλωττιστής C για να έχει την δυνατότητα να μεταγλωττίσει αρχεία.

Η λήψη της γλώσσας στον κεντρικό υπολογιστή θα γίνει μέσω της επίσημης σελίδας της Nim. Αφού ολοκληρωθεί η λήψη, πρέπει να γίνει αποσυμπίεση του αρχείου σε κάποιο φάκελο και η εκτέλεση του προγράμματος finish.exe, το οποίο προσθέτει στην μεταβλητή περιβάλλοντος PATH ότι χρειάζεται για να είναι λειτουργικός ο μεταγλωττιστής της Nim. Στις εικονικές μηχανές η εγκατάσταση θα γίνει γράφοντας την ακόλουθη εντολή στο τερματικό και έπειτα ακολουθώντας τις οδηγίες που θα εμφανιστούν για την τροποποίηση της μεταβλητής περιβάλλοντος PATH:

```
curl https://nim-lang.org/choosenim/init.sh -sSf | sh
```

Για την δημιουργία των εφαρμογών, θα γίνει χρήση βιβλιοθηκών που δεν εγκαθιστούνται μαζί με την γλώσσα. Για την λήψη αυτών θα γίνει χρήση του Nimble. Το Nimble είναι ο προεπιλεγμένος διαχειριστής πακέτων για τη γλώσσα προγραμματισμού Nim. Μπορεί να αναζητήσει πακέτα Nim, να εγκαταστήσει εξαρτήσεις (dependencies), να δημιουργήσει νέα πακέτα και να τα ανεβάσει στην επίσημη λίστα πακέτων. Το Nimble εγκαθιστάται μαζί με την γλώσσα Nim. Το Nimble έχει κάποιες εξαρτήσεις χρόνου εκτέλεσης από εξωτερικά εργαλεία. Αυτά τα εργαλεία χρησιμοποιούνται για τη λήψη πακέτων Nimble. Για παράδειγμα, εάν ένα πακέτο φιλοξενείται στο GitHub, πρέπει να έχει εγκατασταθεί το git και να προστεθεί στη μεταβλητή περιβάλλοντος PATH. Για την λήψη νέων βιβλιοθηκών λοιπόν χρησιμοποιείται η εξής εντολή:

```
nimble install “package name”
```

Όπου package το όνομα της βιβλιοθήκης.

Για να έχουμε πρόσβαση στις λειτουργίες μίας βιβλιοθήκης σε ένα πρόγραμμα Nim, παρόμοια με άλλες γλώσσες προγραμματισμού, πρέπει να την «εισάγουμε» στο πρόγραμμα μέσω της εντολής import “module name”. Παρακάτω θα αναφερθούν οι βιβλιοθήκες που θα χρησιμοποιηθούν για την ανάπτυξη της εφαρμογής (συμπεριλαμβανομένων και αυτών που εγκαθιστούνται μαζί με την γλώσσα Nim), καθώς επίσης και το ποια είναι η λειτουργία τους .

Macros

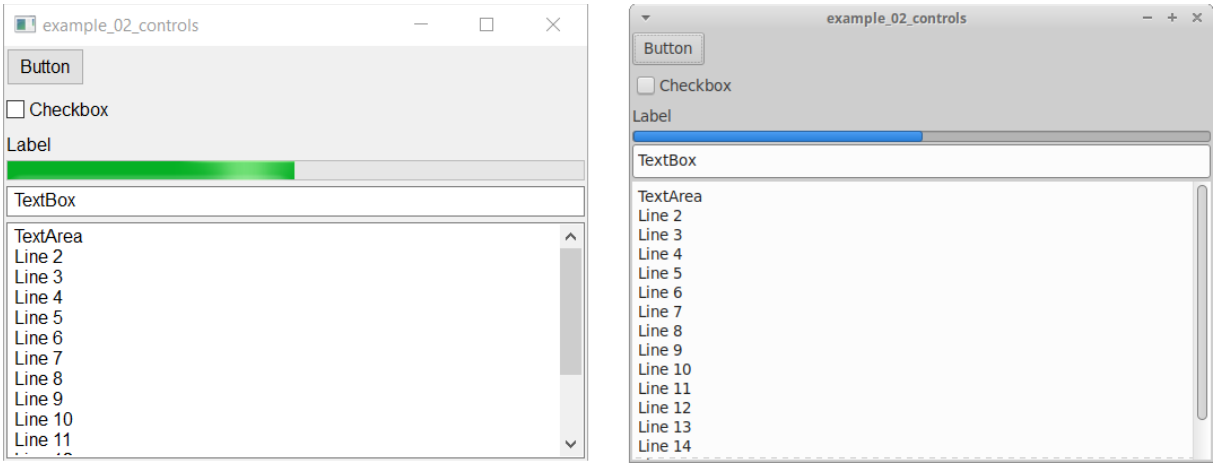
Οι μακροεντολές (macros) είναι μια δυνατότητα μεταπρογραμματισμού της γλώσσας Nim. Ο μεταπρογραμματισμός είναι μια τεχνική προγραμματισμού υπολογιστών κατά την οποία τα προγράμματα έχουν τη δυνατότητα να αντιμετωπίζουν άλλα προγράμματα ως δεδομένα τους. Αυτό σημαίνει ότι ένα πρόγραμμα μπορεί να σχεδιαστεί για να διαβάζει, να δημιουργεί, να αναλύει ή να μετασχηματίζει άλλα προγράμματα, ακόμη και να τροποποιεί τον εαυτό του, ενώ εκτελείται. [8] Ένα macro της Nim συγκεκριμένα είναι μια συνάρτηση που εκτελείται στο χρόνο μεταγλώττισης και μετατρέπει το δέντρο σύνταξης (AST) της Nim σε διαφορετικό δέντρο. Με αυτόν τον τρόπο μπορεί να γράψει κώδικα δυναμικά το οποίο μειώνει κατά συνέπεια τον όγκο του κώδικα και δίνει στον προγραμματιστή περισσότερες δυνατότητες.

Marshal

Η βιβλιοθήκη Marshal περιέχει διαδικασίες για σειριοποίηση και αποσειριοποίηση αυθαίρετων δομών δεδομένων Nim. Η μορφή σειριοποίησης χρησιμοποιεί JSON. Χρησιμοποιείται από την εφαρμογή για την μετατροπή των δομών δεδομένων Nim σε μια μορφή η οποία είναι εύκολα διαχειρισιμή ως μήνυμα του πρωτοκόλλου STOMP.

NiGui

Το NiGui είναι μια βιβλιοθήκη γραφικών διεπαφής χρήστη (GUI) πολλαπλών πλατφορμών γραμμένη σε Nim. Διαθέτει απλά εργαλεία για αλληλεπίδραση και εμφάνιση κειμένου και εικόνας τα οποία είναι προσβάσιμα μέσω κώδικα. Χρησιμοποιείται για την γραφική αναπαράσταση της κατάστασης των δεδομένων ενός κόμβου από το client πρόγραμμα.



Εικόνα 3.9 - Παραδείγματα γραφικού περιβάλλοντος NiGui

rdstdin

Το rdstdin είναι η βιβλιοθήκη που χρησιμοποιείται για την εισαγωγή δεδομένων μέσω του stdin (standard input). Το stdin διαβάζει από την τυπική ροή εισόδου για τη συλλογή δεδομένων. Από προεπιλογή, η τυπική ροή εισόδου είναι συνδεδεμένη στο πληκτρολόγιο. Όταν καλείται μια συνάρτηση ή διαδικασία του rdstdin, το πρόγραμμα περιμένει την εισαγωγή δεδομένων και στη συνέχεια την επεξεργάζεται ανάλογα.

sequtils

Αν και αυτή η λειτουργική μονάδα έχει το seq (sequence) στο όνομά της, υλοποιεί λειτουργίες όχι μόνο για τον τύπο seq, αλλά για τους τρεις ενσωματωμένους τύπους κοντέινερ κάτω από την ομπρέλα openArray: ακολουθίες (sequences), συμβολοσειρές (strings) και πίνακες (arrays). Περιέχει βοηθητικές συναρτήσεις και διαδικασίες για την δημιουργία και τον χειρισμό αυτών των τύπων δεδομένων. Αξίζει να σημειωθεί ότι οι ακολουθίες στην Nim είναι μια λίστα τιμών, αλλά σε αντίθεση με τους πίνακες, το μήκος της ακολουθίας μπορεί να αλλάξει κατά το χρόνο εκτέλεσης. Για αυτόν τον λόγο χρησιμοποιούνται αρκετά για τις ανάγκες της εφαρμογής.

std/algorithm

Η βιβλιοθήκη algorithm παρέχει επίσης περεταίρω λειτουργικότητα γύρω από τους openArray τύπους δεδομένων, προσφέροντας μια γκάμα από πιο εξειδικευμένες διαδικασίες όπως δυαδική αναζήτηση, ταξινόμηση, συγχώνευση και άλλα.

std/math

Μία βιβλιοθήκη που περιέχει διαδικασίες για πιο ειδικές πράξεις πάνω σε αριθμούς. Κάποιες από αυτές είναι η εύρεση τετραγωνικής ρίζας ή παραγωντικού, ο υπολογισμός

τριγωνομετρικών εξισώσεων ή ακόμα και η εύρεση μέγιστου κοινού διαιρέτη/ελάχιστου κοινού πολλαπλάσιου μίας ομάδας αριθμών.

std/monotimes

Η βιβλιοθήκη std/monotimes υλοποιεί μονοτονικές χρονικές σημάνσεις. Μια μονοτονική χρονική σήμανση αντιπροσωπεύει το χρόνο που έχει περάσει από κάποιο χρονικό σημείο που έχει οριστεί από το σύστημα. Ο τύπος MonoTime αποθηκεύει τη χρονική σήμανση σε ανάλυση νανοδευτερόλεπτου, αλλά η πραγματική υποστηριζόμενη χρονική ανάλυση διαφέρει για διαφορετικά συστήματα. Μία από τις περιπτώσεις χρήσης της είναι, για παράδειγμα, για την καταγραφή του χρόνου εκτέλεσης μιας συνθήκης ή συνάρτησης.

std/net

Αυτή η βιβλιοθήκη υλοποιεί μια διεπαφή υποδοχών (sockets) υψηλού επιπέδου πολλαπλών πλατφορμών. Μια υποδοχή δικτύου είναι μια δομή λογισμικού που χρησιμεύει ως τελικό σημείο για την αποστολή και λήψη δεδομένων μέσω του δικτύου. Μέσω των υποδοχών που παρέχονται από την βιβλιοθήκη net δηλαδή γίνεται δυνατή η σύνδεση μεταξύ της εφαρμογής και του RabbitMQ server.

std/os

Αυτή η βιβλιοθήκη παρέχει μια διεπαφή για βασικές λειτουργίες και χαρακτηριστικά του λειτουργικού συστήματος, όπως ανάκτηση μεταβλητών περιβάλλοντος, εργασίες με καταλόγους και αρχεία, εκτέλεση εντολών shell κ.λπ.

std/random

Η τυπική γεννήτρια τυχαίων αριθμών (RNG) της γλώσσας Nim. Με την χρήση της γίνεται η αρχικοποίηση των δεδομένων των κόμβων με τυχαίες τιμές. Για να παράγει η βιβλιοθήκη τις ίδιες ακριβώς τυχαίες τιμές κάθε φορά χρησιμοποιείται το ίδιο seed. Σε κάποια λειτουργικά συστήματα που δεν παρέχουν την GetRandom συνάρτηση, για να χρησιμοποιηθεί αυτή η βιβλιοθήκη, κατά την μεταγλώττιση του προγράμματος πρέπει να δοθεί η οδηγία `-d:nimNoGetRandom` ως παράμετρος.

stomp

Η βιβλιοθήκη που υλοποιεί τις λειτουργίες του πρωτοκόλλου STOMP. Μέσω αυτής γίνονται οι αποστολές και οι παραλαβές των μηνυμάτων που διαχειρίζεται ο RabbitMQ server.

strutils

Αυτή η λειτουργική μονάδα συστήματος ορίζει κάποιες συνηθισμένες λειτουργίες για την εργασία με συμβολοσειρές, όπως μετατροπή άλλων τύπων δεδομένων σε συμβολοσειρές, συνένωση συμβολοσειρών και άλλες.

sysinfo

Το sysinfo είναι μια βιβλιοθήκη πολλαπλών πλατφορμών για την εύρεση πληροφοριών σχετικά με το λειτουργικό σύστημα ή το υλικό του υπολογιστή. Οι πληροφορίες περιλαμβάνουν μεταξύ άλλων την έκδοση του λειτουργικού συστήματος, την ποσότητα της ελεύθερης μνήμης και το όνομα του επεξεργαστή.

system/io

Μία λειτουργική μονάδα προσφέρει έναν τρόπο διαχείρισης αρχείων. Με την χρήση της μπορούμε να διαβάσουμε, να διαγράψουμε, να τροποποιήσουμε ή να δημιουργήσουμε καινούρια αρχεία. Μετά την έκδοση Nim 2.0 αυτή η βιβλιοθήκη είναι πλέον μέρος της γλώσσας και δεν χρειάζεται εισαγωγή.

threadpool

Η βιβλιοθήκη για την χρήση νημάτων (threads) στην γλώσσα Nim ονομάζεται threadpool. Τα threads είναι ο τρόπος με τον οποίων εντολές σε ένα πρόγραμμα μπορούν να εκτελεστούν παράλληλα. Είναι χρήσιμα για την δημιουργία διαδραστικών εφαρμογών, οπού θέλουμε οι οποιεσδήποτε εργασίες εκτελούνται στο παρασκήνιο να μην επηρεάζουν την αποκριτικότητα της εφαρμογής. Έχουν επίσης την δυνατότητα να εκτελέσουν μια σειρά από υπολογισμούς πιο γρήγορα απ' ότι θα μπορούσε μια σειριακή εκτέλεση. Πέρα από την threadpool η Nim έχει πλέον πιο εξειδικευμένες βιβλιοθήκες για νήματα όπως οι malebolgia, taskpools και weave, αλλά για τις ανάγκες της εφαρμογής η threadpool είναι αρκετή.

3.4 Υλοποίηση του συστήματος

3.4.1 Σκοπός της εφαρμογής

Όπως έχει αναφερθεί και νωρίτερα, η εφαρμογή που θα υλοποιηθεί είναι μια πλατφόρμα υπολογισμών μεγάλης κλίμακας, της οποίας η λειτουργικότητα θα εξεταστεί μέσω της υλοποίησης ενός κατανεμημένου Game of Life (GOL). Το Game of Life, είναι ένα κυψελιδικό αυτόματο που επινοήθηκε από τον Βρετανό μαθηματικό John Horton Conway το 1970. [9] Ένα κυτταρικό ή κυψελιδικό αυτόματο είναι ένα υπολογιστικό μοντέλο συστημάτων με αναδυόμενη πολυπλοκότητα που μελετάται στη θεωρία των αυτομάτων.

Όπως και ο λογικός χώρος GridWorld, ένα κυψελικό αυτόματο αποτελείται από ένα πλέγμα κυψελών, η κάθε μια σε μία από έναν πεπερασμένο αριθμό καταστάσεων, όπως ενεργοποίηση και απενεργοποίηση. Το πλέγμα μπορεί επίσης να είναι σε οποιονδήποτε πεπερασμένο αριθμό διαστάσεων. Ένα κυψελιδικό αυτόματο λειτουργεί με τα εξής βήματα:

1. Για κάθε κελί, ορίζεται ένα σύνολο κελιών που ονομάζεται γειτονιά του.

2. Μια αρχική κατάσταση (χρονική στιγμή t = 0) επιλέγεται εκχωρώντας μια κατάσταση για κάθε κελί.

3. Δημιουργείται μια νέα γενιά (προχωρώντας το t κατά 1), σύμφωνα με κάποιον σταθερό κανόνα (συνήθως, μια μαθηματική συνάρτηση) που καθορίζει τη νέα κατάσταση κάθε κελιού σε σχέση με την τρέχουσα κατάσταση του κελιού και τις καταστάσεις των κελιών στη γειτονιά του.

4. Το βήμα 3 επαναλαμβάνεται μέχρι τον τερματισμό του προγράμματος. [10]

Το κυψελιδικό αυτόματο Game of Life συγκεκριμένα, είναι ένα δισδιάστατο ορθογώνιο πλέγμα από τετράγωνα κελιά, καθένα από τα οποία βρίσκεται σε μία από τις δύο πιθανές καταστάσεις, ζωντανή ή νεκρή (ή κατοικημένη και μη κατοικημένη, αντίστοιχα). Κάθε κελί αλληλοεπιδρά με τους οκτώ γείτονές του, που είναι τα κελιά που είναι οριζόντια, κάθετα ή διαγώνια γειτονικά. Η μαθηματική σχέση με την οποία υπολογίζεται η επόμενη κατάσταση κάθε κελιού είναι η εξής:

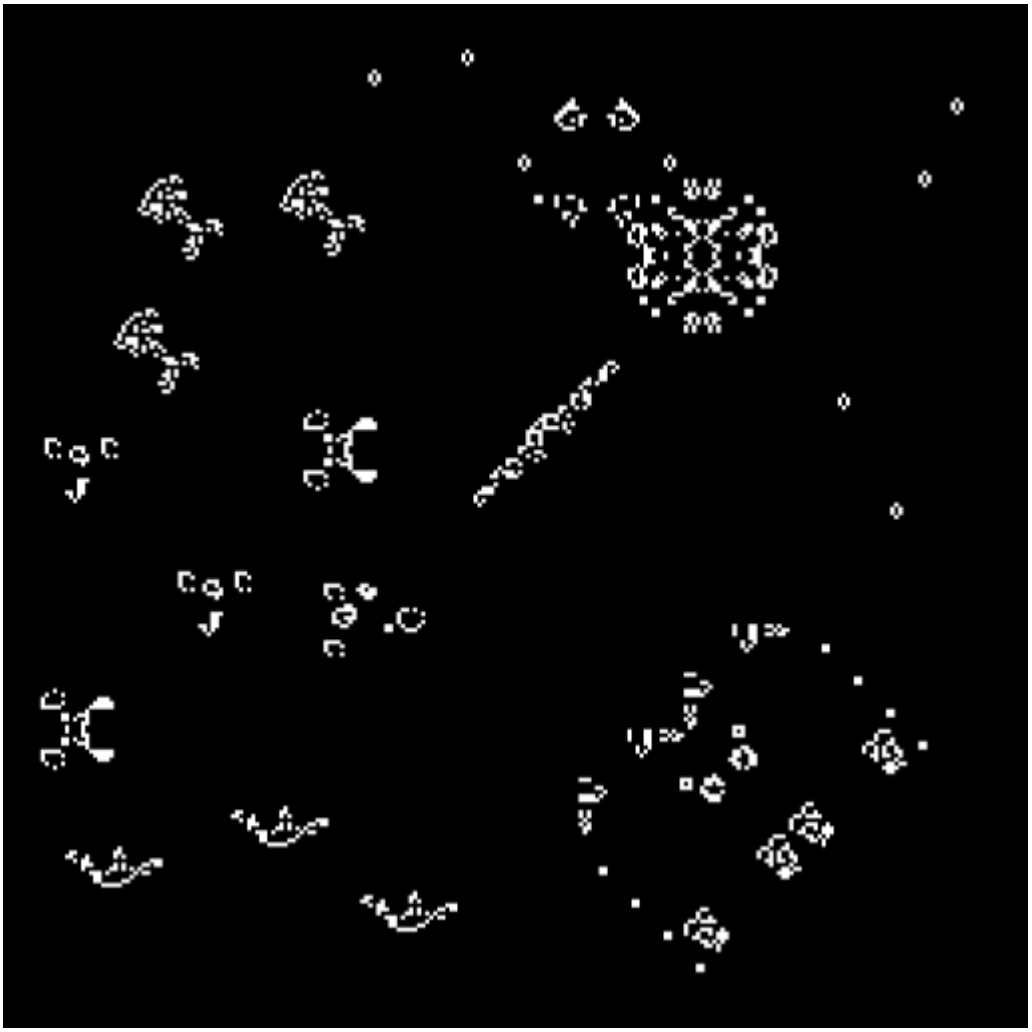
1. Κάθε ζωντανό κύτταρο με λιγότερους από δύο ζωντανούς γείτονες πεθαίνει, σαν από υποπληθυσμό.

2. Οποιοδήποτε ζωντανό κύτταρο με δύο ή τρεις ζωντανούς γείτονες ζει στην επόμενη γενιά.

3. Κάθε ζωντανό κύτταρο με περισσότερους από τρεις ζωντανούς γείτονες πεθαίνει, σαν από υπερπληθυσμό.

4. Κάθε νεκρό κύτταρο με ακριβώς τρεις ζωντανούς γείτονες γίνεται ζωντανό κύτταρο, σαν με αναπαραγωγή.

Η αρχική κατάσταση αποτελεί τον σπόρο (seed) του συστήματος. Η πρώτη γενιά δημιουργείται εφαρμόζοντας τους παραπάνω κανόνες ταυτόχρονα σε κάθε κυψέλη του πλέγματος, ζωντανό ή νεκρό. γεννήσεις και θάνατοι συμβαίνουν ταυτόχρονα, και η διακριτή στιγμή που συμβαίνει αυτό ονομάζεται τικ.



Εικόνα 3.10 - Παράδειγμα κατάστασης κόσμου Game Of Life

Παρακάτω θα αναφερθεί πως υλοποιείται κάθε επιμέρους πρόγραμμα του συστήματος, με αναφορά στις κύριες διαδικασίες και την λειτουργία τους. Ο κώδικας για τις διαδικασίες που θα μιλήσουμε, μαζί με οποιαδήποτε άλλα κομμάτια του κώδικα, είναι διαθέσιμος για ανάγνωση στο παράρτημα Α.

3.4.2 Master node

Εδώ περιγράφονται οι διαδικασίες που περιέχονται μόνο στο master node. Οι διαδικασίες που αποτελούν μέρος του master και των slave nodes εξίσου θα αναφερθούν παρακάτω.

initMessage

Η διαδικασία που καλείται όταν το master node δέχεται το μήνυμα για την αρχικοποίηση του λογικού χώρου μέσω του gateway. Κάνει την απλή εργασία της αποθήκευσης μεταβλητών όπως την διάσταση του χώρου, τον αριθμό των στρωμάτων (layers) των μεταβλητών GridWorld που θα κρατούνται στην μνήμη κ.λπ. Ένα στρώμα μεταβλητών είναι η κατάστασή του συνόλου του λογικού χώρου σε μία χρονική στιγμή. Λόγω του ότι η επόμενη κατάσταση του λογικού χώρου είναι κάθε στιγμή άμεσα εμπλεκόμενη με την προηγούμενή της χρειαζόμαστε τουλάχιστον 2 στρώματα.

specMessage

Καλείται όταν το master node λαμβάνει μηνύματα από τα slave nodes για την ποσότητα της ελεύθερης μνήμης τους, το όνομα του επεξεργαστή τους και το αποτέλεσμα από τον αλγόριθμο μέτρησης υπολογιστικής ισχύος. Αυτές οι πληροφορίες θα χρησιμοποιηθούν αργότερα για τον υπολογισμό του μεγέθους των slices που θα λάβει κάθε node.

getCPUScores

Η διαδικασία που καλείται για να βρεθούν και να αποθηκευτούν οι βαθμολογίες των επεξεργαστών των nodes. Το αρχείο που περιέχει τις βαθμολογίες για πάνω από 4.500 επεξεργαστές σαρώνεται σειριακά και προσπαθεί να ταιριάξει το όνομα του επεξεργαστή της τρέχον γραμμής με κάθε έναν από την λίστα των επεξεργαστών των nodes. Όταν επιτύχει η βαθμολογία αποθηκεύεται.

memArrange

Η κύρια διαδικασία που ξεχωρίζει τα υπόλοιπα nodes από το master node σε θέμα λειτουργικότητας. Η memArrange χρησιμοποιείται για να υπολογιστεί πόσα nodes χρειάζονται για την αναπαράστασή του λογικού χώρου GridWorld και ποιο θα είναι το μέγεθος των slices που θα ανατεθεί σε κάθε node, καθώς επίσης και την διάταξη των nodes στον λογικό χώρο. Η διαδικασία αυτή κάνει τις παρακάτω κύριες εργασίες με τη σειρά:

1. Ξεκινάει έχοντας ως ενεργό node μόνο το master node

2. Αποθηκεύει το ποσοστό της συνολικής επεξεργαστικής ισχύς που κατέχει το κάθε ενεργό node σε μια ακολουθία (sequence)

3. Ο λογικός χώρος των δεδομένων χωρίζεται σε περίπου ίσα slices και δίνονται στα ενεργά nodes

4. Τα nodes τακτοποιούνται στον δισδιάστατο χώρο με βάση την ελεύθερη μνήμη.

5. Δημιουργείται μια ακολουθία που περιέχει την ποσότητα της υπολειπόμενης μνήμης κάθε ενεργού node, ταξινομημένης με φθίνουσα σειρά κατά την υπολογιστική ισχύ.

6. Αφαιρούνται σειρές του slice από τα nodes που δεν έχουν ελεύθερη υπολειπόμενη μνήμη, από αυτά που έχουν μέγεθος slice που δεν μπορεί να υποστηρίξει το STOMP πρωτόκολλο ως φορτίο, και από αυτά που προκαλούν τα γειτονικά nodes τους να μην έχουν ελεύθερη μνήμη με το να είναι οι μεγαλύτεροι σε μέγεθος γείτονες από τους οποίους πρέπει να πάρουν δεδομένα. Οι σειρές αυτές προστίθενται στα nodes που έχουν αρκετή ελεύθερη μνήμη, δίνοντας προτεραιότητα σε αυτά που έχουν μεγαλύτερη επεξεργαστική ισχύ μέσω τις ακολουθίας που δημιουργήθηκε στο βήμα 5.

7. Τέλος, αν μετά από αυτές τις εργασίες υπάρχει node με αρνητική ελεύθερη μνήμη, τότε γίνεται ενεργό ένα ακόμη node και η εκτέλεση συνεχίζεται από το βήμα 2.

simpleArrange

Η διαδικασία που καλείται αν έχει αποφασιστεί ήδη από τον client ο αριθμός των nodes που θα χρησιμοποιηθούν για την προσομοίωση GOL. Τα δεδομένα χωρίζονται απλά σε ίσα slices και τα nodes κατατάσσονται στον χώρο με την σειρά.

getNeighboursToCheck

Χρησιμοποιείται από την memArrange για να βρεθεί για ποιους γείτονες του είναι ένα node ο μεγαλύτερος σε μέγεθος γείτονας, δηλαδή αυτός με το μεγαλύτερο slice. Η διαδικασία αυτή έχει μια επιπλέον παράμετρο για το αν θα πρέπει να είναι ένα node ο μοναδικός μεγαλύτερος γείτονας. Θέλουμε να είναι μοναδικός όταν ελέγχουμε για το αν πρέπει να αφαιρέσουμε σειρές από το slice του, και μοναδικός ή μη-μοναδικός όταν επιχειρούμε να του προσθέσουμε σειρές.

createPartitionTable

Η διαδικασία που φτιάχνει το partition table που αποστέλλεται και χρησιμοποιείται από το gateway πρόγραμμα για την δρομολόγηση συγκεκριμένων μηνυμάτων στα σωστά nodes.

findFirstIt

Ένα template για την εύρεση του πρώτου αντικειμένου σε μία ακολουθία, το οποίο ικανοποιεί μια συνθήκη που του δίνεται ως παράμετρος. Τα templates είναι ένας απλός μηχανισμός αντικατάστασης που λειτουργεί στα αφηρημένα συντακτικά δέντρα της Nim. Τα templates επεξεργάζονται στο σημασιολογικό πέρασμα του μεταγλωττιστή. [11]

3.4.3 Κοινές διαδικασίες μεταξύ Slave nodes και Master node

unrollLoops

Το macro που υλοποιεί το κόσκινο του Ερατοσθένη για την υπολογισμό της επεξεργαστικής ισχύος ενός node με βάση τον χρόνο εκτέλεσής του. Όπως αναφέρει και το όνομα του, κάνει χρήση μιας τεχνικής που αποκαλείται loop unrolling ή ξετύλιγμα βρόχων. Το loop unrolling είναι μια τεχνική μετασχηματισμού βρόχου που προσπαθεί να βελτιστοποιήσει την ταχύτητα εκτέλεσης ενός προγράμματος σε βάρος του μεγέθους του αρχείου του. Ο στόχος του ξετυλίγματος βρόχου είναι να αυξηθεί η ταχύτητα ενός προγράμματος μειώνοντας ή εξαλείφοντας τις εντολές που ελέγχουν τον βρόχο, όπως η αριθμητική των δεικτών και οι δοκιμές "τέλος βρόχου" σε κάθε επανάληψη, [12] μειώνοντας τις ποινές διακλάδωσης, καθώς και αποκρύβοντας καθυστερήσεις, συμπεριλαμβανομένης της καθυστέρησης στην ανάγνωση δεδομένων από τη μνήμη. [13] Για να εξαλειφθεί αυτή η υπολογιστική επιβάρυνση, οι βρόχοι μπορούν να ξαναγραφούν ως επαναλαμβανόμενη ακολουθία παρόμοιων ανεξάρτητων εντολών. [14] Αυτό ακριβώς είναι η εργασία που εκτελεί στην ουσία η μακροεντολή unrollLoops για να ξεδιπλώσει τους βρόχους. Ο κώδικας αυτός λόγω της φύσης των μακροεντολών μετατρέπεται κατά την μεταγλώττιση από 70 σε 1943 σειρές κώδικα που περιέχουν 32 διαφορετικές εκδοχές βρόχων που έχουν ξετυλιχτεί. Τέλος ο κώδικας που δημιουργείται από την παραπάνω μακροεντολή δουλεύει ως εξής:

1. Οι βασικοί πρώτοι έχουν μειωθεί μόνο σε περιττές τιμές.

2. Ο αποκοπή (culling) μεταξύ των byte σημαίνει ότι η αποκοπή είναι σε ζυγές τιμές, καθώς τα byte έχουν οκτώ bit.

3. Κάνοντας αυτό, εμφανίζεται ένα επαναλαμβανόμενο μοτίβο modulo όπου η ίδια θέση bit θα επισημαίνεται για κάθε οκτώ culll και το byte που έχει καλυφθεί θα είναι ακριβώς η ‘βασική τιμή πρώτου αριθμού’ μακριά από την τελευταία φορά που αυτή η θέση bit επισημάνθηκε ως αποκομμένη.

4. Τα παραπάνω σημαίνουν ότι κάθε επαναλαμβανόμενο μοτίβο για μια δεδομένη βασική τιμή πρώτου αριθμού διαφέρει μόνο από τη θέση bit από την οποία ξεκινά, για οκτώ ενδεχόμενα.

5. Λόγω των μαθηματικών modulo, υπάρχουν μόνο διαφορετικά μοτίβα για τις βασικές τιμές πρώτων αριθμών modulo οκτώ, και δεδομένου ότι όλες οι

βασικές τιμές πρώτων αριθμών είναι περιττές σημαίνει ότι το λιγότερο σημαντικό bit είναι πάντα ένα. Υπάρχουν μόνο τέσσερις εκδοχές μοτίβου αποκοπής. Αυτά τα τέσσερα μοτίβα επί τις οκτώ πιθανές θέσεις εκκίνησης σημαίνει ότι υπάρχουν μόνο 32 συνολικά μοτίβα αποκοπής.

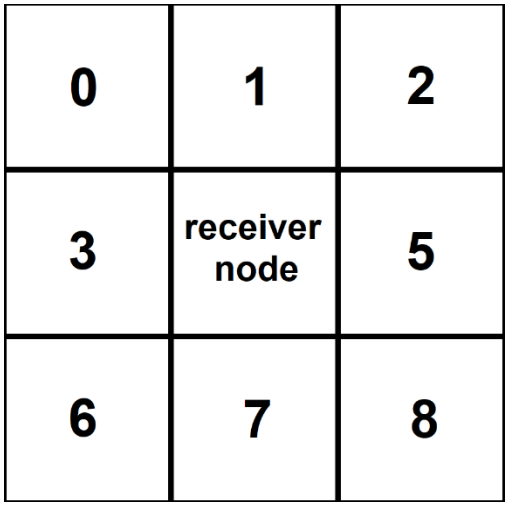
6. Επειδή καθένα από αυτά τα πιθανά μοτίβα αποκοπής είναι σταθερά για ένα δεδομένο μοτίβο, μπορεί κανείς να χρησιμοποιήσει σταθερές (άμεσες) τιμές κάλυψης για τις θέσεις απομάκρυνσης οκτώ μοτίβων, που σημαίνει ότι δεν είναι απαραίτητο το "bit twiddling".

7. Για ένα δεδομένο πιθανό μοτίβο αποκοπής για μια δεδομένη βασική τιμή πρώτου αριθμού, κάθε δείκτης byte είναι μια σταθερή μετατόπιση που αναπτύχθηκε από τη βασική αρχική τιμή από τον πρώτο δείκτη byte στα οκτώ μοτίβα, πράγμα που σημαίνει ότι εάν κάποιος αναγκάσει τον μεταγλωττιστή να χρησιμοποιήσει ευρετηρίαση καταχωρητή από βασικό καταχωρητή (που κάνουν αυτόματα οι πιο αποτελεσματικοί μεταγλωττιστές τύπου C), εάν είναι διαθέσιμοι οκτώ καταχωρητές, τότε ολόκληρος ο βρόχος μπορεί να γίνει χρησιμοποιώντας λειτουργίες καταχωρητή και εάν υπάρχουν άλλοι δύο καταχωρητές διαθέσιμοι για να κρατηθούν τη βάση η αρχική τιμή και η οριακή τιμή λήξης βρόχου, τότε ολόκληρη η αποκοπή για ένα δεδομένο εύρος μπορεί να γίνει εξ ολοκλήρου χρησιμοποιώντας λειτουργίες καταχωρητή διαφορετικές από την πράξη ανάγνωσης/τροποποίησης/εγγραφής που στην πραγματικότητα επισημαίνει ένα κομμάτι ως αποκομμένο.

8. Η παραπάνω τεχνική λειτουργεί αποτελεσματικά μόνο για μικρότερες βασικές τιμές πρώτων αριθμών όπου το εύρος αποκοπής δεν υπερβαίνει τον αριθμό των byte στη μνήμη cache της CPU L1, και στην πραγματικότητα λιγότερο από αυτό, καθώς υπάρχει σημαντικός χρόνος εγκατάστασης για τον βρόχο και επίσης απώλεια χρόνου για το επιπλέον σφάλμα πρόβλεψης διακλάδωσης που προκαλείται από την υπολογισμένη επιλογή μοτίβων ενός από τα 32 μοτίβα, καθώς αυτή η επιλογή φαίνεται να είναι τυχαία για την CPU και επομένως δεν μπορεί να προβλεφθεί αποτελεσματικά. [15]

dataMessageLoop

Η διαδικασία που καλείται όταν ένα node λαμβάνει μήνυμα από κάποιο από τα γειτονικά nodes του ή από το gateway για τροποποίηση/διάβασμα ενός συγκεκριμένου cell. Στο σώμα του μηνύματος βρίσκονται τα δεδομένα, και στις κεφαλίδες πληροφορίες όπως η θέση του node στον λογικό χώρο, τον τύπο του μηνύματος κ.τ.λ. Αν είναι μήνυμα για τροποποίηση/διάβασμα cell τότε το τροποποιεί ή στέλνει την τιμή του στον client αντίστοιχα. Αλλιώς συνεχίζει επεξεργάζοντας το μήνυμα του γειτονικού node, με το να μετατρέπει τις δύο τιμές της απόλυτης θέση του στο καρτεσιανό πλέγμα (x,y) σε μια μόνο τιμή. Η τιμή αυτή υπολογίζεται ως η θέση του node αποστολέα σε σχέση με την θέση του node παραλήπτη όπως φαίνεται στην εικόνα:

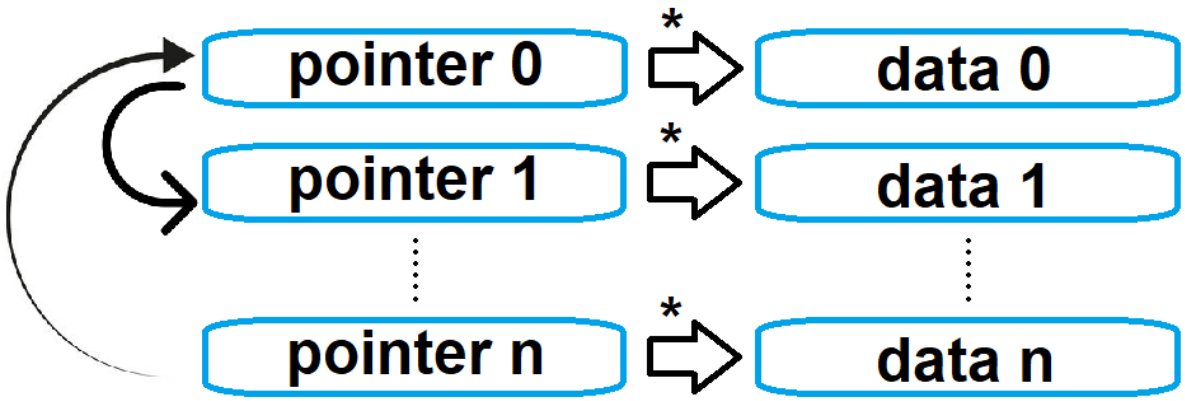


Εικόνα 3.11 - Σχετική θέση γειτόνων ενός node

Με τη βοήθεια της σχετικής θέσης που υπολογίστηκε, το κομμάτι των δεδομένων που χρειάζεται το node παραλήπτης, συλλέγεται από το μήνυμα του γειτονικού node και αποθηκεύεται στην αντίστοιχη θέση στην ακολουθία (sequence). Αν με το τρέχον μήνυμα ο παραλήπτης έχει πλέον λάβει μηνύματα από όλους τους γείτονές του για αυτήν την γενιά (λογικό βήμα), συνεχίζει με το να υπολογίσει και να σώσει την επόμενη κατάσταση του slice του, και αφού το κάνει ανανεώνει τον πίνακα δεικτών των στρωμάτων των δεδομένων και προχωράει στο επόμενο λογικό βήμα. Τέλος, η νέα του κατάσταση αποστέλλεται στα γειτονικά nodes και αποθηκεύεται επίσης σε αρχείο στον δίσκο, για να χρησιμοποιηθεί σε περίπτωση που το πρόγραμμα του node τερματιστεί απροσδόκητα για οποιονδήποτε λόγο. Αξίζει να σημειωθεί ότι κάθε μήνυμα που στέλνεται από τα nodes έχει την οδηγία `prefetch-count 1`, που σημαίνει ότι κάθε τέτοιο μήνυμα μπορεί να παραληφθεί ένα τη φορά πριν γίνει αναγνώριση (ack). Αυτό βοηθάει στο να μην επιχειρήσει να λάβει κάποιο node μήνυμα ενώ δεν έχει αρκετή μνήμη για να το διαχειριστεί.

rearrangeLayers

Η διαδικασία που αναλαμβάνει την ανασύνταξη της ακολουθίας των δεικτών των στρωμάτων των δεδομένων. Οι δείκτες αυτοί αποθηκεύουν την θέση μνήμης για κάθε στρώμα δεδομένων του slice και μετατοπίζονται σε κάθε λογικό βήμα. Χρησιμοποιούνται αντί να τροποποιούμε άμεσα τα δεδομένα διότι με αυτόν τον τρόπο η χρήση της μεταβλητής temp για την μετατόπιση των στοιχείων της ακολουθίας είναι μικρότερου μεγέθους, όντας απλά μια θέση μνήμης σε αντίθεση με την άμεση μέθοδο που θα χρειαζόταν στην ουσία μνήμη αρκετή για ένα επιπλέον slice.



Εικόνα 3.12 - Αναπαράσταση της λειτουργίας της rearrangeLayers και της ακολουθίας των δεικτών

calculateNextStep

Μέσω της calculateNextStep γίνεται ο υπολογισμός της νέας κατάστασης του slice με βάση τους κανόνες του μαθηματικού μοντέλου, που στην παρούσα περίπτωση είναι το Game Of Life. Ο υπολογισμός μπορεί να γίνει γρήγορα με μια μόνο εντολή που ελέγχει και τους 4 κανόνες του game of life:

```
nextStep[i][j] = (aliveNeighbours or currentStep[i][j]) == 3)
```

Όπου aliveNeighbours είναι οι ζωντανοί γείτονες για το τρέχον cell. Το or στην συγκεκριμένη περίπτωση είναι η πράξη bitwise OR. Το bitwise OR είναι μια δυαδική πράξη που παίρνει δύο μοτίβα bit ίδιου μήκους και εκτελεί τη λογικά περιεκτική πράξη OR σε κάθε ζεύγος αντίστοιχων bit.

calculateBorders

Η διαδικασία που καλείται από την calculateNextStep για τον υπολογισμό της κατάστασης των ακριανών δεδομένων του slice. Τα δεδομένα που «συνορεύουν» στην ουσία με δεδομένα από slices άλλων nodes πρέπει να λάβουν υπόψιν τις τιμές αυτές, και όχι μόνο τις τιμές των γειτόνων που βρίσκονται στο ίδιο slice με αυτά. Αυτό το κάνουν μέσω της ακολουθίας των δεδομένων των γειτόνων του node που ανανεώνεται με κάθε καινούριο γειτονικό μήνυμα.

drawGrid

Η drawGrid καλείται στο τέλος κάθε λογικού βήματος και εκτυπώνει στο τερματικό την κατάσταση των δεδομένων. Τα ζωντανά cells εμφανίζονται με τον χαρακτήρα ‘0’.

3.4.4 Gateway

nodeDataMessage

Με την χρήση αυτής της διαδικασίας λαμβάνονται τα δεδομένα για την αρχικοποίηση του λογικού χώρου (αριθμός στρωμάτων, μέγεθος χώρου κ.τ.λ.) από τον client και αποστέλλονται στον master node, οπού θα επεξεργαστούν ανάλογα.

infoMessage

Καλείται όταν το master node στέλνει το partition table και σχήμα του λογικού χώρου των slices. Το partition table αποθηκεύεται για χρήση ενώ το σχήμα του λογικού χώρου στέλνεται στον client.

messageLoop

Η διαδικασία με την βοήθεια της οποίας γίνεται η επικοινωνία μεταξύ των nodes και του client αφού γίνει η ρύθμιση της εφαρμογής με τη βοήθεια του master node. Η διαδικασία επιλέγει τι να κάνει με το μήνυμα μέσω της κεφαλίδας messageType η οποία παίρνει 4 διαφορετικές τιμές που σημαίνουν τα εξής:

1: Το μήνυμα είναι ένα slice και προέρχεται από κάποιο node. Πρέπει να αποσταλεί στον client για εμφάνιση στο UI.

2: Το μήνυμα προέρχεται από τον client και αναθέτει στο gateway πρόγραμμα το node στο οποίο πρέπει να εγγραφεί για να λαμβάνει μηνύματα του, τα οποία θα στέλνονται με τη σειρά τους πίσω στον client

3: Το μήνυμα προέρχεται από τον client και χρησιμοποιείται για να διαβαστεί ένα συγκεκριμένο κελί από τον λογικό χώρο. Το gateway πρόγραμμα χρησιμοποιεί το partition table για να δρομολογήσει το μήνυμα στο σωστό node, το οποίο περιέχει το επιθυμητό κελί.

4: Το μήνυμα προέρχεται από τον client και χρησιμοποιείται για να τροποποιηθεί ένα συγκεκριμένο κελί από τον λογικό χώρο.

3.4.5 Client

initWorld

Η διαδικασία από την οποία ο χρήστης εισάγει τις παραμέτρους του λογικού χωρου μέσω πληκτρολογίου στο τερματικό. Τα δεδομένα αυτά έπειτα στέλνονται στο gateway πρόγραμμα.

readCell και writeCell

Οι διαδικασίες που καλούνται όταν επιθυμούμε να διαβάσουμε ή να τροποποιήσουμε ένα συγκεκριμένο κελί αντίστοιχα. Το μήνυμα με το ανάλογο messageType αποστέλλεται στο gateway πρόγραμμα μαζί με την απόλυτη θέση του κελιού στο πλέγμα.

drawMenu

Μέσω αυτής της διαδικασίας το πρόγραμμα δημιουργεί την γραφική διεπαφή που ο χρήστης θα χρησιμοποιήσει για να επιλέξει το node από το οποίο επιθυμεί να παρακολουθήσει τα δεδομένα. Πρέπει να τρέχει σε ξεχωριστό thread από την διαδικασία που περιμένει για λήψη μηνυμάτων, για να μπορούν να λειτουργήσουν ταυτόχρονα.

nodeSelected

Καλείται όταν ο χρήστης επιλέγει το node του οποίου τα δεδομένα θέλει να παρατηρήσει. Η απόλυτη θέση του node συλλέγεται από το GUI και αποστέλλεται στο gateway πρόγραμμα. Επίσης ανοίγει το παράθυρο που θα φιλοξενήσει τα δεδομένα του slice.

drawGraphics

Η διαδικασία που δημιουργεί το παράθυρο που θα παρέχει την γραφική αναπαράσταση των δεδομένων του slice. Κάθε κελί είναι ένα pixel στον καμβά του παραθύρου. Τα πράσινα pixel είναι ζωντανά κελιά και τα γκρι νεκρά.

messageLoop

Η messageLoop είναι υπεύθυνη για την διαχείριση τον μηνυμάτων των nodes. Όταν λαμβάνει μήνυμα που προήλθε λόγο αίτησης διαβάσματος συγκεκριμένου κελιού, τότε εμφανίζει την τιμή του στο τερματικό. Αν το μήνυμα είναι δεδομένα slice τροποποιεί το αντικείμενο της εικόνας που περιέχει τον καμβά με τα νέα δεδομένα και την ανανεώνει.

Κεφάλαιο 4ο: Αποτελέσματα Εφαρμογής

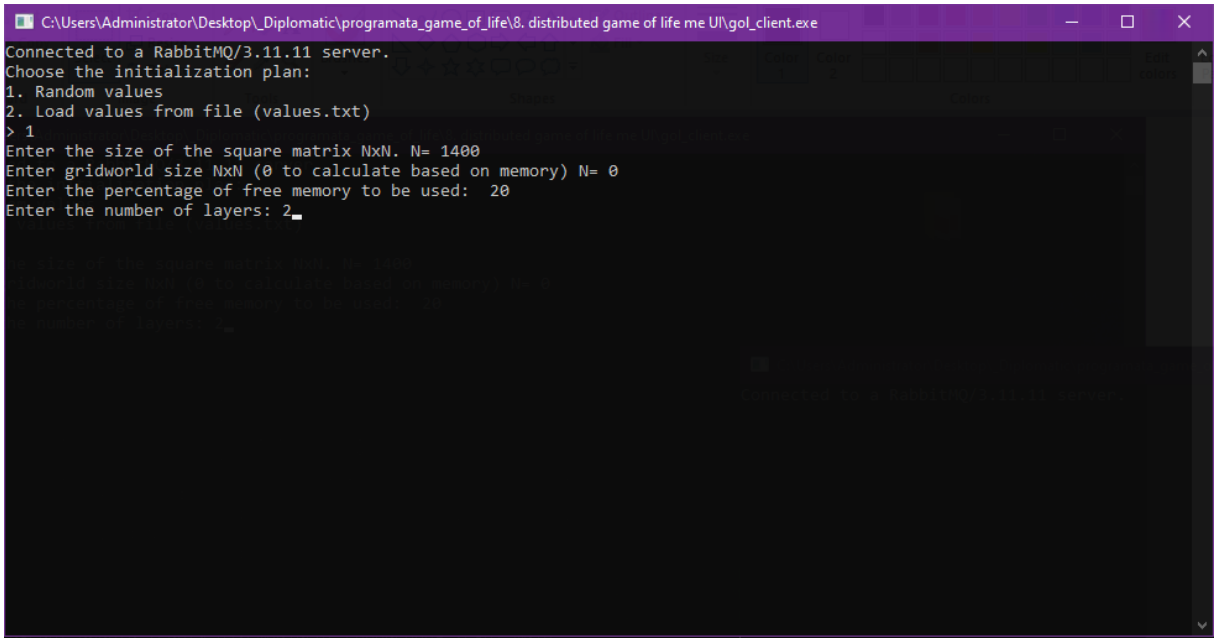
4.1 Ροή εκτέλεσης της εφαρμογής

Πριν γίνει δυνατό το να εκτελεστούν τα προγράμματα, πρέπει πρώτα να μεταγλωττιστούν. Αυτό γίνεται με την εντολή:

```
nim compile “path”/”name” -d:release
```

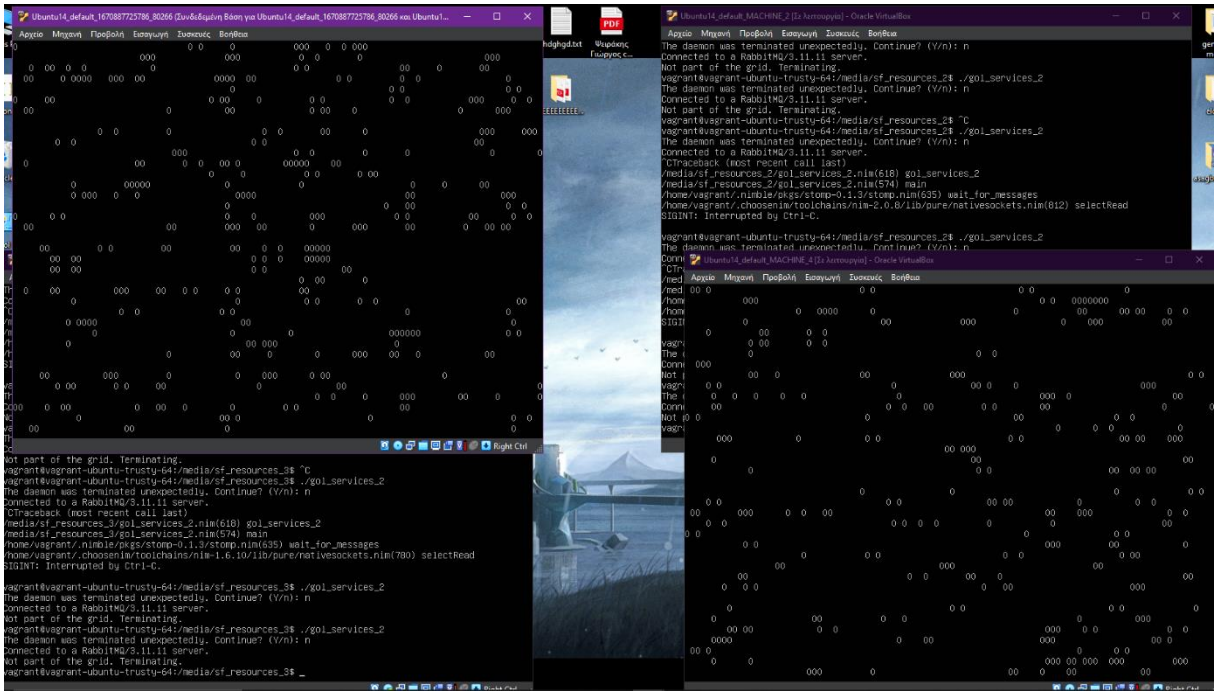
Η παράμετρος d:release χρησιμοποιείται για να δημιουργήσει ο μεταγλωττιστής γρηγορότερο κώδικα μηχανής.

Παρακάτω θα παρουσιαστούν τα αποτελέσματα της εφαρμογής με στιγμιότυπα οθόνης που πάρθηκαν κατά την εκτέλεση του προγράμματος. Αρχικά εκκινούμε όλα τα προγράμματα (client, gateway, nodes, master node).



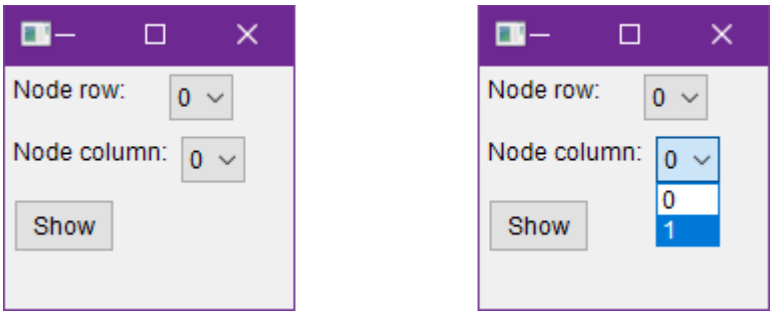
Εικόνα 4.1 - Αρχικοποίηση των παραμέτρων του λογικού χώρου

Εισάγουμε τις παραμέτρους στον client, με τις οποίες θα γίνει αρχικοποίηση του κόσμου GridWorld. Αυτές θα περαστούν με τη βοήθεια του gateway στο master node για να γίνει με την χρήση τους ο διαμοιρασμός των slices στα nodes του δικτύου.



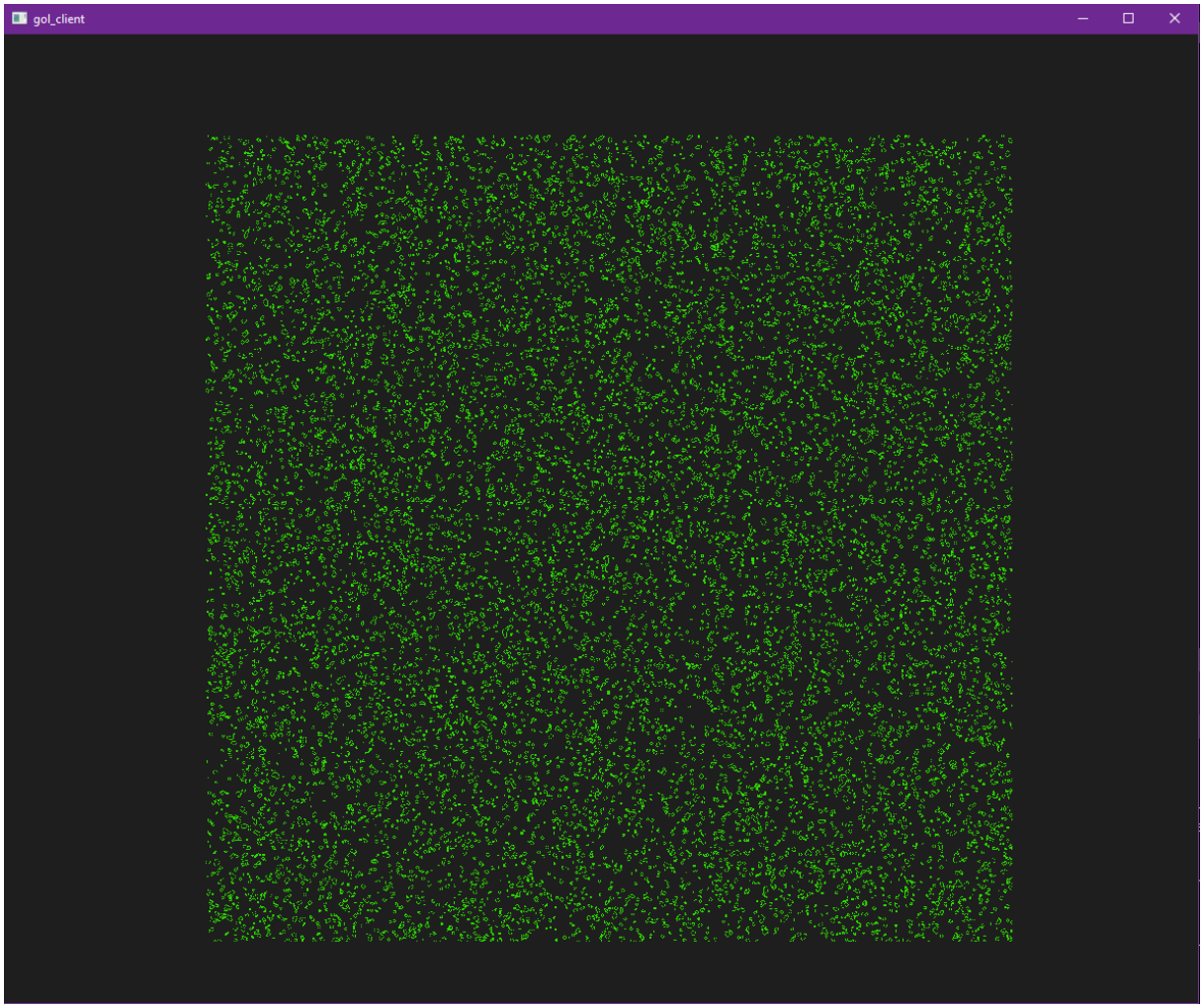
Εικόνα 4.2 - Εργασία των nodes

Στη συνέχεια τα nodes που αποφάσισε το master node ότι πρέπει να ενεργοποιηθούν, αρχίζουν να εκτελούν το πρόγραμμα Game Of Life. Υπολογίζουν τα slices τους για κάθε λογικό βήμα και το αποστέλλουν στον RabbitMQ server για δρομολόγηση στους γείτονες τους. Τα προγράμματα των nodes που δεν διαχειρίζονται μέρος του λογικού χώρου τερματίζεται αυτόματα.



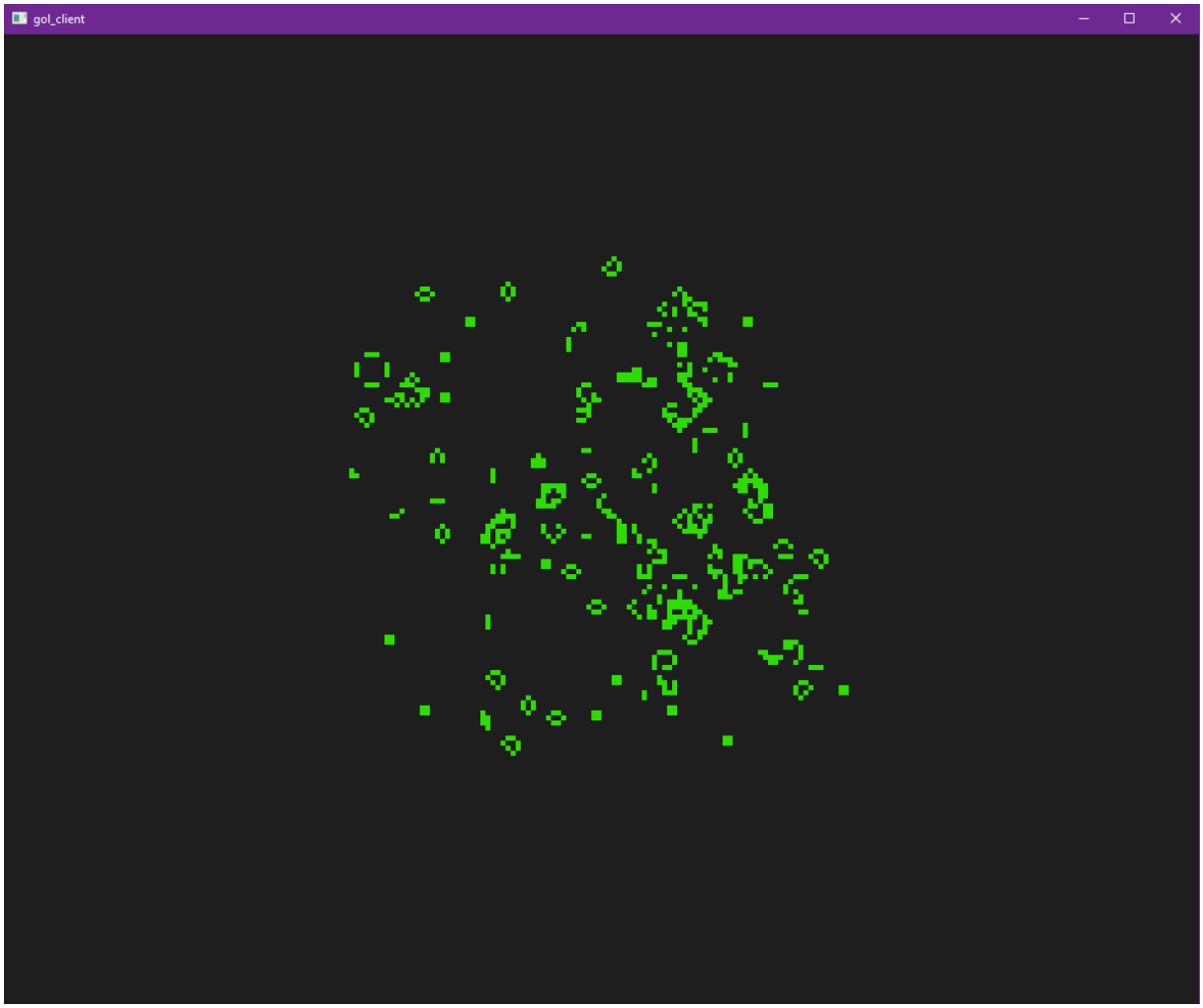
Εικόνα 4.3 - Μενού επιλογής node

Μετά την εκκίνηση του προγράμματος Game Of Life, το μενού για την επιλογή node για παρακολούθηση γίνεται λειτουργικό. Μέσω αυτού επιλέγεται το node βάση της απόλυτης θέσης του στον χώρο (γραμμή και στήλη) και εμφανίζεται η γραφική αναπαράσταση των δεδομένων του με το κουμπί Show.



Εικόνα 4.4 - Το slice ενός node με γραφικά

Αφού επιλεχθεί το node, εμφανίζεται ο καμβάς στον οποίον χρωματίζονται πράσινα τα ζωντανά κελιά του slice. Ο καμβάς φυσικά ανανεώνεται κάθε φορά που το node περνάει στο επόμενο λογικό βήμα. Αν επιθυμούμε να δούμε κάποιο άλλο node αρκεί απλά να κλείσουμε αυτό το παράθυρο και να επιλέξουμε το επόμενο node από το μενού.



Εικόνα 4.5 – Ένα μικρότερου μεγέθους slice με πιο γνώριμα μοτίβα

Παραπάνω εμφανίζεται ένα slice με μοτίβα που είναι συνηθισμένο να δημιουργηθούν σε ένα κυψελιδικό αυτόματο Game Ofof Life έπειτα από έναν αριθμό λογικών βημάτων εκτέλεσης. Τα δεδομένα του slice έχουν μεγεθυνθεί ουτος ώστε να είναι πιο εύκολα αναγνωρίσιμα τα μοτίβα.

4.2 Συμπεράσματα και προτάσεις βελτίωσης

Η πλατφόρμα μπορεί να χρησιμοποιηθεί σε προβλήματα (κατανεμημένης) τεχνητής νοημοσύνης, όπως συστήματα ευφυών πρακτόρων, έξυπνων δικτύων, έξυπνων σπιτιών κλπ, όπως και σε εφαρμογές υπολιγιστικής επιστήμης στην βιολογία και την ιατρική. Ακόμα και με το γεγονός ότι η εφαρμογή αυτή δοκιμάστηκε μόνο τοπικά, προσομοιώνοντας ένα δίκτυο υπολογιστών με την βοήθεια εικονικών μηχανών, είναι φανερό το ότι οι δυνατότητες της στην διεκπεραίωση επιστημονικών υπολογισμών μεγάλης κλίμακας είναι απεριόριστες. Δεδομένου του ότι τα ίδια ακριβώς προγράμματα μπορούν να φιλοξενηθούν από ένα δίκτυο πολλές τάξεις μεγέθους μεγαλύτερο, ο συνολικός όγκος δεδομένων που μπορεί να

διαχειριστεί η εφαρμογή είναι φαινομενικά απεριόριστος και στην πραγματικότητα κρίνεται κυρίως από τις δυνατότητες του RabbitMQ server ή του αντίστοιχου messaging broker που έχει επιλεχτεί. Υπάρχουν όμως κάποια πράγματα που θα μπορούσαν να βελτιωθούν. Αρχικά τα nodes της εφαρμογής δεν συγχρονίζονται στο ίδιο λογικό βήμα δημιουργίας slices με αποτέλεσμα τα δεδομένα να μην είναι πλήρως έγκυρα. Μία εφαρμογή επιστημονικών υπολογισμών επιβάλλεται να παράγει σωστά αποτελέσματα για να είναι χρήσιμη. Ο συγχρονισμός αυτός θα μπορούσε να επιτευχθεί μέσω της δημιουργίας ενός πρωτοκόλλου για τον συγχρονισμό των nodes πριν την δημιουργία της αρχικής κατάστασης, και κατα την περάτωση ενός υπολογιστικού βήματος. Επιπλέον, η εφαρμογή θα μπορούσε να υλοποιεί ένα πιο περίπλοκο μαθηματικό μοντέλο όπως πχ για την πρόβλεψη καιρού με συλλογή μεταβλητών κατάστασης της ατμόσφαιρας από διάφορα αισθητήρια, ή ένα μοντέλο πρόβλεψης βλάστησης. Τέλος, θα μπορούσε να υλοποιηθεί ένα σύστημα όπου ανενεργά nodes προσθέτονται δυναμικά αν κάποιο αποσυνδεθεί από το δίκτυο για οποιοδήποτε λόγο, για να εξασφαλίζεται έτσι η διαθεσιμότητα του συστήματος.

Η γλώσσα Nim που χρησιμοποιήθηκε για την υλοποίηση της εφαρμογής, είναι μια γλώσσα με πολλές δυνατότητες και με ταχύτητα που είναι παρόμοια με αυτήν της C, όπως επαληθεύτηκε και με την εκτέλεση προγραμμάτων benchmark και συγκρίνοντας τους χρόνους εκτέλεσης. Είναι επίσης πραγματικά δυνατό να δουλέψει με πολλαπλές πλατφόρμες χωρίς αλλαγές στον κώδικα. Η μόνη εξαίρεση εμφανίζεται όταν ο κώδικας χρησιμοποιεί μια βιβλιοθήκη φτιαγμένη από εθελοντές η οποία δεν έχει σχεδιαστεί για να δουλεύει με πολλαπλές πλατφόρμες. Επίσης, λόγο του ότι η γλώσσα Nim δεν είναι ιδιαίτερα γνωστή σε σχέση με δημοφιλής γλώσσες όπως Python, Java κ.τ.λ. η εύρεση της κατάλληλης βιβλιοθήκης για κάποια εργασία ίσως να μην είναι δυνατή και αυτό ίσως αποθαρρύνει νέους προγραμματιστές από το να την προτιμήσουν.

Για την υλοποίηση της εφαρμογής, χρησιμοποίησα τεχνολογίες που δεν είχα ξαναχρησιμοποιήσει και απέκτησα επίγνωση του πώς δουλεύουν τα συστήματα επεξεργασίας δεδομένων μεγάλης κλίμακας. Πιο συγκεκριμένα, έμαθα πώς να χρησιμοποιώ και να δημιουργώ εικονικές μηχανές. Έμαθα να προγραμματίζω στην γλώσσα Nim, η οποία έχει πολλές δυνατότητες. Έμαθα για την έννοια του μεταπρογραμματισμού και το πώς μπορούμε να τον επιτύχουμε μέσω των ισχυρών εργαλείων που παρέχει η Nim γνωστών ως μακροεντολές. Έμαθα επίσης για την ύπαρξη των message brokers και πως servers όπως ο RabbitMQ και η Kafka δουλεύουν, όπως επίσης και το πώς συνεργάζονται με πρωτόκολλα όπως το STOMP και το AMQP για να διακινήσουν μηνύματα. Τέλος, χρησιμοποίησα για πρώτη φορά προγραμματισμό με threads και έμαθα πώς διαφέρει από την ασύγχρονη εκτέλεση. Όλες αυτές οι γνώσεις που αποκόμισα κάνουν αυτήν την εργασία επιτυχία, παρά το γεγονός ότι υπάρχουν και άλλες βελτιώσεις που μπορούν να γίνουν μελλοντικά.

Βιβλιογραφία

[1] G. H. Golub και M. J. Ortega, Scientific Computing and Differential Equations – An Introduction to Numerical Methods, Elsevier Science, 1992.

[2] P. Staab, «Scientific Computing,» [Ηλεκτρονικό]. Available: https://math.libretexts.org/Bookshelves/Scientific_Computing_Simulations_and_Modeling/Scientific_Computing_(Staab).

[3] New Mexico State University, «MPI Parallelization,» [Ηλεκτρονικό]. Available: https://hpc.nmsu.edu/discovery/mpi/introduction/.

[4] L. Johansson, «RabbitMQ for beginners,» [Ηλεκτρονικό]. Available: https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html.

[5] Broadcom, «AMQP 0-9-1 Model Explained,» Broadcom, [Ηλεκτρονικό]. Available: https://www.rabbitmq.com/tutorials/amqp-concepts.

[6] Apache Kafka Documentation. [Ηλεκτρονικό]. Available: https://kafka.apache.org/documentation/.

[7] vmware, «Virtual Machine,» [Ηλεκτρονικό]. Available: https://www.vmware.com/topics/virtual-machine.

[8] K. Czarnecki και U. Eisenecker, σε *Generative Programming: Methods, Tools, and Applications*.

[9] M. Gardner, «The fantastic combinations of John Conway's new solitaire game 'life',» *Scientific American*, 1970.

[10] T. Toffoli και N. Margolus, Cellular Automata Machines: A New Environment for Modeling, MIT Press, 1987 .

[11] A. Rumpf, «Nim Tutorial (Part II),» [Ηλεκτρονικό]. Available: https://nim-lang.org/docs/tut2.html.

[12] A. V. Aho και J. D. Ullman, Principles of compiler design, Addison-Wesley Pub. Co., 1977.

[13] W. P. Petersen και P. Arbenz, Introduction to Parallel Computing.

[14] A. Nicolau, Loop Quantization: Unwinding for Fine-Grain Parallelism Exploitation, Cornell University, Dept. of Computer Science, Ithaca, N.Y., 1985.

[15] G. W. Goodsman, «Benchmarking the Beast,» 28 July 2021. [Ηλεκτρονικό]. Available: https://nim-lang.org/blog/2021/07/28/Nim-Efficient-Expressive-Elegant-Benchmarking.html.

Παράρτημα Α

Σε αυτό το παράρτημα αναφέρονται τα κομμάτια του κώδικα που χρησιμοποιήθηκαν.

Gol_main_3.nim (master node)

```
#!nimrod


import
   std/[net],
   stomp,
   strutils,
   sequtils,
   marshal,
   std/math,
   std/algorithm,
   std/os,
   rdstdin,
   std/random,
   sysinfo,
   std/monotimes,
   macros


when NimMajor < 2:
  import system/io


from times import inMilliseconds


type Partition = object
  rowStart: int
  rowEnd: int
  columnStart: int
  columnEnd: int


var matrixLength: int
```

```
var totalCellNum: int


var username, password, address: string
var node_position_row, node_position_column: int
var nodes_length_row, nodes_length_column: int
var nodeDataLengthSelf: seq[int]


var memNeeded: int


var neighbourNodes: int = 0
var totalNodes: int = 1
var nodesParsed: int = 0
var nodesNeeded: int = 1


const STOMP_PAYLOAD_SIZE = 4 * 1024 * 1024
const cushion = 0.95
const nodeAllowedSize = int(cushion*STOMP_PAYLOAD_SIZE)


var allowedMemoryUsage: float = 0.6      # 1 means 100% of free memory
var nodeMemory: seq[float]
var isSimpleDist = false
var isRandom = true
var selfIndex = 0


var wasInterrupted: bool = false


var nodeCpus: seq[string]
var nodePositions: seq[ tuple[row:int, column:int]]
var nodePositions_new: seq[ tuple[row:int, column:int]]
var nodeCpuBench: seq[float]


var activeNodesPerRow: seq[int]
var nodeDataLength: seq[tuple[rows:int, columns:int]]
var nodeWeightedPerformance: seq[float]
var memorySorted: seq[tuple[memory:float, nodeIndex:int]]
var memoryLeftSortedBySpec: seq[tuple[memory:float, specs:float, nodeIndex:int]]
var insufficientMemory: bool
```

```
var localRowLength: int
var localColumnLength: int

var matrixData: seq[int]
var neighbourData: seq[seq[int]]

var gameField: seq[seq[seq[int]]]
var gameLayers: int
var gameStep: int

var partitionTable: seq[seq[Partition]]
var cellValueChange: tuple[layer, row, column, value:int] = (-1, -1, -1, -1)

var layerPointers: seq[ptr seq[seq[int]]]

#--------------------------------------------------------------------------BENCHMARK-----
-------------------------------------------------------------------------------------------
-----
const cLOOPS {.intdefine.} = 1225

# avoids some "bit-twiddling" for better speed...
const cBITMSK = [ 1'u8, 2, 4, 8, 16, 32, 64, 128 ]

macro unrollLoops(ca, sz, strtndx, bp: untyped) =
  let cmpstsalmtid = newIdentNode("cmpstsalmt")
  let szbitsid = newIdentNode("szbits")
  let strtndx0id = newIdentNode("strtndx0")
  let strt0id = newIdentNode("strt0")
  let strt7id = newIdentNode("strt7")
  let endalmtid = newIdentNode("endalmt")
  let bpintid = newIdentNode("bpint")
  let cullaid = newIdentNode("culla")
  result = quote do:
    let `szbitsid` = `sz` shl 3
    let `cmpstsalmtid` = `ca` + `sz`
    let `bpintid` = `bp`.int
    let `strtndx0id` = `strtndx`
```

```
  let `strt0id` = `strtndx0id` shr 3
for i in 1 .. 7:
  let strtndxido = newIdentNode("strtndx" & $(i - 1))
  let strtndxidn = newIdentNode("strtndx" & $i)
  let strtid = newIdentNode("strt" & $i)
  result.add quote do:
    let `strtndxidn` = `strtndxido` + `bp`
    let `strtid` = (`strtndxidn` shr 3) - `strt0id`
let csstmnt = quote do:
  case (((`bpintid` and 0x6) shl 2) + (`strtndx0id` and 7)).uint8
  of 0'u8: break
csstmnt.del 1 # delete last dummy "of"
for n in 0'u8 .. 0x3F'u8: # actually used cases...
  let pn = (n shr 2) or 1'u8
  let cn = n and 7'u8
  let mod0id = newLit(cn)
  let cptr0id = "cptr0".newIdentNode
  let loopstmnts = nnkStmtList.newTree()
  for i in 0'u8 .. 7'u8:
    let mskid = newLit(1'u8 shl ((cn + pn * i.uint8) and 7).int)
    let cptrid = ("cptr" & $i).newIdentNode
    let strtid = ("strt" & $i).newIdentNode
    if i == 0'u8:
      loopstmnts.add quote do:
        let `cptrid` = cast[ptr uint8](`cullaid`)
    else:
      loopstmnts.add quote do:
        let `cptrid` = cast[ptr uint8](`cullaid` + `strtid`)
    loopstmnts.add quote do:
      `cptrid`[] = `cptrid`[] or `mskid`
  loopstmnts.add quote do:
    `cullaid` += `bpintid`
  let ofbrstmnts = quote do:
    while `cullaid` < `endalmtid`:
      `loopstmnts`
    `cullaid` = ((`cullaid` - `ca`) shl 3) or `mod0id`.int
    while `cullaid` < `szbitsid`:
```

```
        let `cptr0id` = cast[ptr uint8](`ca` + (`cullaid` shr 3))
        `cptr0id`[] = `cptr0id`[] or cBITMSK[`cullaid` and 7]
        `cullaid` += `bpintid`
    csstmnt.add nnkOfBranch.newTree(
      newLit(n),
      ofbrstmnts
    )
  for n in 0x40'u8 .. 255'u8: # fill in defaults for remaining possibilities
    csstmnt.add nnkOfBranch.newTree(
      newLit(n),
      nnkStmtList.newTree(
        nnkBreakStmt.newTree(
          newEmptyNode()
        )
      )
    )
  result.add quote do:
    let `endalmtid` = `cmpstsalmtid` - `strt7id`
    var `cullaid` = `ca` + `strt0id`
    `csstmnt`
#  echo csstmnt[9].astGenRepr # see AST for a given case
#  echo csstmnt[9].toStrLit # see code for a given case
#  echo result.toStrLit # see entire produced code at compile time

proc benchSoE(): iterator(): int {.closure.} =
  var cmpsts = newSeq[byte](16384)
  let cmpstsa = cast[int](cmpsts[0].addr)
  for _ in 0 ..< cLOOPS:
    for i in 0 .. 254: # cull to square root of limit
      if (cmpsts[i shr 3] and cBITMSK[i and 7]) == 0'u8: # if prime -> cull its composites
        let bp = i +% i +% 3
        let swi = (i +% i) *% (i +% 3) +% 3
        unrollLoops(cmpstsa, 16384, swi, bp)
  return iterator(): int {.closure.} =
    yield 2 # the only even prime
    for i in 0 .. 131071: # separate iteration over results
      if (cmpsts[i shr 3] and cBITMSK[i and 7]) == 0'u8:
```

```
        yield i +% i +% 3
#--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
-----------------


proc generateValues(rows, columns: int): seq[int] =
  #var result = newSeq[int](rows*columns)
  var seed=0
  var value=0

  for i in 0 ..< result.len:

    seed = rand(100)
    value = (int)seed>86
    result[i] = value

  #result

proc toSingleDimension(seqtoedit: seq[seq[int]]): seq[int] =
  #var result = newSeq[int]()
  for i in 0 ..< seqtoedit.len:
    for j in 0 ..< seqtoedit[i].len:
      result.add(seqtoedit[i][j])
  #result

proc getSeqSlice(sequence: seq[seq[int]], rowStart, rowEnd, columnStart, columnEnd: int):
seq[seq[int]] =
  for i in 0 ..< sequence.len:
      var seqToAdd = newSeq[int]()
      for j in 0 ..< sequence[i].len:
        if rowStart <= i and rowEnd >= i and columnStart <= j and columnEnd >= j:
          seqToAdd.add(sequence[i][j])

      if seqToAdd.len > 0:
        result.add(seqToAdd)



proc rearrangeLayers() =
```

```
  var tempPtr = layerPointers[gameLayers-1]
  for i in countdown(layerPointers.len-1,1):
    layerPointers[i] = layerPointers[i-1]
  layerPointers[0] = tempPtr



proc initialize() =

  randomize()

  nodesParsed = 0
  neighbourData = newSeqWith(9, newSeq[int](1))

  let creds = open("./credentials.txt")
  defer: creds.close()

  node_position_row = parseInt(creds.readLine())
  node_position_column = parseInt(creds.readLine())
  nodes_length_row = parseInt(creds.readLine())
  nodes_length_column = parseInt(creds.readLine())
  address = creds.readLine()
  username = creds.readLine()
  password = creds.readLine()

  totalNodes = nodes_length_row*nodes_length_column

  nodeCpus = newSeq[string](totalNodes)
  nodeMemory = newSeq[float](totalNodes)

  nodeCpuBench = newSeq[float](totalNodes)

  if fileExists("./positions.txt"):

    var continueStep = readLineFromStdin("The daemon was terminated unexpectedly. Continue?
(Y/n): ")
    if continueStep.toUpperAscii != "Y": return

    wasInterrupted = true
```

```
    let positions = open("./positions.txt")
    defer: positions.close()
    node_position_row = parseInt(positions.readLine())
    node_position_column = parseInt(positions.readLine())
    nodeDataLengthSelf = to[seq[int]](positions.readLine())
    activeNodesPerRow = to[seq[int]](positions.readLine())
    gameLayers = parseInt(positions.readLine())
    if fileExists("./selfData.txt"):
      let selfDataFile  = open("selfData.txt")
      defer: selfDataFile.close()
      var selfData = to[seq[int]](selfDataFile.readLine())
      gameField = newSeq[seq[seq[int]]](gameLayers)

      var nodeData = selfData.distribute(nodeDataLengthSelf[0], false)
      layerPointers = newSeq[ptr seq[seq[int]]](gameLayers)
      for i in 0 ..< gameLayers:
        gameField[i] = nodeData
        layerPointers[i] = addr gameField[0]



  if fileExists("./parsed.txt"):
    let parsedFile = open("parsed.txt")
    defer: parsedFile.close()
    nodesParsed = parseInt(parsedFile.readLine())
    gameStep = parseInt(parsedFile.readLine())



  for i in 0 .. 8:
    if fileExists($i & "data.txt"):
      let dataFile = open($i & "data.txt")
      defer: dataFile.close()
      var data = to[seq[int]](dataFile.readLine())
      neighbourData[i] = data

proc benchCPU() =
  let strt = getMonotime()
```

```
  let answr = benchSoE()
  var cnt = 0; for _ in answr(): cnt += 1
  var elpsd = float((getMonotime() - strt).inMilliseconds)
  elpsd = 100/elpsd

  nodeCpuBench[0] = elpsd

proc prepareForMessages() =

  nodePositions = newSeq[ tuple[row:int, column:int]](totalNodes)
  memorySorted = newSeq[tuple[memory:float, nodeIndex:int]](totalNodes)
  activeNodesPerRow = newSeq[int]()
  nodeDataLength = newSeq[tuple[rows:int, columns:int]]()
  nodeWeightedPerformance = newSeq[float]()

  #nodeDataLength.add((0,0)) #$$
  #nodeWeightedPerformance.add(0.0) #$$
  #activeNodesPerRow.add(0) #$$

  nodePositions[0] = (node_position_row, node_position_column)

  benchCPU()
  nodesParsed += 1



proc cleanScreen*() =
     write(stdout,"\e[H\e[J")

template findFirstIt*(s: seq, pred: untyped): untyped =

  var result = -1
  var index = 0
  for it {.inject.} in items(s):
    if pred:
      result = index
      break
    inc index
```

```
  result

proc fillGrid(gameField: var seq[seq[int]], rows, columns: int) =
    var seed=0
    var value=0
    for i in 0 ..< rows:
        for j in 0 ..< columns:
          seed = rand(100)
          value = (int)seed>85
          gameField[i][j] = value

proc calculateNeighbours(i,j: int, gameField: seq[seq[int]]): int =
    var neighbours = 0;
    for x in max(0, i - 1) .. min(i + 1, localRowLength-1):
        for y in max(0, j - 1) .. min(j + 1, localColumnLength-1):
            if x != i or y != j: neighbours += gameField[x][y]

    result= neighbours;

proc calculateBorders(currentStep, nextStep: var seq[seq[int]]) =
  for i in 0 ..< nodeDataLengthSelf[0]:
    var lastRowElem = nodeDataLengthSelf[1]-1
    var aliveNeighboursL = calculateNeighbours(i, 0, currentStep)
    var aliveNeighboursR = calculateNeighbours(i, lastRowElem, currentStep)

    if i == 0:
      aliveNeighboursL += neighbourData[0][0]
      aliveNeighboursL += neighbourData[1][0]
      aliveNeighboursL += neighbourData[1][1]
      aliveNeighboursR += neighbourData[2][0]
      aliveNeighboursR += neighbourData[1][lastRowElem-1]
      aliveNeighboursR += neighbourData[1][lastRowElem]

    if i == nodeDataLengthSelf[0]-1:
      aliveNeighboursL += neighbourData[6][0]
      aliveNeighboursL += neighbourData[7][0]
      aliveNeighboursL += neighbourData[7][1]
```

```
      aliveNeighboursR += neighbourData[8][0]
      aliveNeighboursR += neighbourData[7][lastRowElem-1]
      aliveNeighboursR += neighbourData[7][lastRowElem]
    else:
      aliveNeighboursL += neighbourData[3][i+1]
      aliveNeighboursR += neighbourData[5][i+1]

    aliveNeighboursL += neighbourData[3][i]
    aliveNeighboursR += neighbourData[5][i]

    nextStep[i][0] = (int)((aliveNeighboursL or currentStep[i][0]) == 3)
     nextStep[i][lastRowElem] = (int)((aliveNeighboursR or currentStep[i][lastRowElem]) ==
3)

  for j in 1 ..< nodeDataLengthSelf[1]-1:
    var lastColumnElem = nodeDataLengthSelf[0]-1
    var aliveNeighboursU = calculateNeighbours(0, j, currentStep)
    var aliveNeighboursD = calculateNeighbours(lastColumnElem, j, currentStep)
    aliveNeighboursU += neighbourData[1][j] + neighbourData[1][j+1]
    aliveNeighboursD += neighbourData[7][j] + neighbourData[7][j+1]
    nextStep[0][j] = (int)((aliveNeighboursU or currentStep[0][j]) == 3)
                nextStep[lastColumnElem][j]       =       (int)((aliveNeighboursD       or
currentStep[lastColumnElem][j]) == 3)

proc calculateNextStep(currentStep, nextStep: var seq[seq[int]], isDistributed: bool) =
    #this will be different in the daemons
    var rows = nodeDataLengthSelf[0]-(int)isDistributed
    var columns = nodeDataLengthSelf[1]-(int)isDistributed
    var start = (int)isDistributed

    if isDistributed:
      calculateBorders(currentStep,nextStep)

    for i in start ..< rows:
      for j in start ..< columns:
        var aliveNeighbours = calculateNeighbours(i, j, currentStep)
        nextStep[i][j] = (int)((aliveNeighbours or currentStep[i][j]) == 3)
```

```
proc padNeighbours() =
  for i in [0,2,6,8]:
    if neighbourData[i].len == 0: neighbourData[i].add(0)

  while neighbourData[3].len < nodeDataLengthSelf[0]:
    neighbourData[3].add(0)
  while neighbourData[5].len < nodeDataLengthSelf[0]:
    neighbourData[5].add(0)

  while neighbourData[1].len < nodeDataLengthSelf[1]:
    neighbourData[1].add(0)
  while neighbourData[7].len < nodeDataLengthSelf[1]:
    neighbourData[7].add(0)

proc drawGrid(gameField: seq[seq[int]]) =
  cleanScreen()
  for i in 0 ..< gameField.len: #localRowLength[0]:
     for j in 0 ..< gameField[i].len: #localColumnLength[1]:
         var cellState = if (bool)gameField[i][j]: "0" else: " "
         stdout.write cellState

     echo " "
  echo " "
  echo "Step " & $gameStep



# Announce when we're connected.
proc connected( c: StompClient, r: StompResponse ) =
   echo "Connected to a ", c["server"], " server."



#NETWORK STEP 4: received spec message
proc specMessage( c: StompClient, r: StompResponse ) =

  sleep(500)
  let id = r[ "ack" ]
  var nodeMem: float
```

```
  var cpuBench: float
  var pos_row: int
  var pos_column: int
  var cpu: string

  try:
    nodeMem = parseFloat(r["mem"])
    cpuBench = parseFloat(r["bench"])
    pos_row = parseInt(r["pos-row"])
    pos_column = parseInt(r["pos-column"])
    cpu = r.payload
  except CatchableError as e:
    let
      e = getCurrentException()
      msg = getCurrentExceptionMsg()
    echo "Got exception ", repr(e), " with message ", msg
    c.nack( id )
    return

  nodeCpus[nodesParsed] = cpu
  nodeCpuBench[nodesParsed] = cpuBench
  nodeMemory[nodesParsed] = nodeMem
  nodePositions[nodesParsed] = (pos_row , pos_column)

  nodesParsed += 1
  c.ack( id )

proc initMessage( c: StompClient, r: StompResponse ) =

  let id = r[ "ack" ]

  try:
    matrixLength = parseInt(r["matrix-len"])
    totalCellNum = matrixLength*matrixLength
    allowedMemoryUsage = parseFloat(r["mem-use"])
    nodesNeeded = parseInt(r["node-num"])
    gameLayers = parseInt(r["layers"])
```

```
    nodesNeeded *= nodesNeeded
    matrixData = to[seq[int]](r.payload)
    isSimpleDist = nodesNeeded > 0
    isRandom = matrixData.len == 0

    if nodesNeeded > totalNodes:
       echo "unable to initialize with " & $nodesNeeded & " nodes. Not enough nodes.
Terminating"
      c.ack(id)
      system.quit()

  except CatchableError as e:
    let
      e = getCurrentException()
      msg = getCurrentExceptionMsg()
    echo "Got exception ", repr(e), " with message ", msg
    c.nack( id )
    return

  c.ack( id )

proc dataMessageLoop( c: StompClient, r: StompResponse ) =

  sleep(1000)
  let id = r[ "ack" ]

  let msgType = parseInt(r["type"])
  if msgType == 3:
    let cellPos_row = parseInt(r["cell-row"])
    let cellPos_col = parseInt(r["cell-col"])
    let layer = parseInt(r["layer"])
    var headers: seq[ tuple[name:string, value:string] ]
    headers.add( ("type", $3) )
       c.send(  "/topic/clientmessage",  $layerPointers[layer][][cellPos_row][cellPos_col],
"text/plain", headers )
    c.ack( id )
    return
  if msgType == 4:
```

```
  let cellPos_row = parseInt(r["cell-row"])
  let cellPos_col = parseInt(r["cell-col"])
  let layer = parseInt(r["layer"])
  var valueToSet = parseInt(r.payload)
  cellValueChange = (layer, cellPos_row, cellPos_col, valueToSet)
  c.ack( id )
  return

#let data_rows = parseInt(r["data-rows"])




try:
  let rowActual = parseInt(r["pos-row"])
  let columnActual = parseInt(r["pos-column"])

  #if rowActual == node_position_row and columnActual == node_position_column: return

  let relativePosition_row = rowActual-node_position_row+1
  let relativePosition_column = columnActual-node_position_column+1
  let singleRelativeIndex = relativePosition_column + 3*relativePosition_row
  var nodeData = to[seq[int]](r.payload).distribute(nodeDataLengthSelf[0], false)
  #echo $nodeDataLengthSelf[0]
  #echo $nodeDataLengthSelf[1]
  #echo $nodeData.len
  #echo $nodeData[0].len
  #var asd = readLineFromStdin("test ")
  neighbourData[singleRelativeIndex] = @[]
  var datavalue: int
  if 3 <= singleRelativeIndex and singleRelativeIndex <= 5:

    for i in 0 ..< min(nodeData.len,nodeDataLengthSelf[0]):
      datavalue = nodeData[i][(nodeData[i].len-1)*(int)(singleRelativeIndex==3)]
      neighbourData[singleRelativeIndex].add(datavalue)
  elif singleRelativeIndex mod 3 == 1:
    for i in 0 ..< min(nodeData[0].len,nodeDataLengthSelf[1]):
      datavalue = nodeData[(nodeData.len-1)*(int)(singleRelativeIndex==1)][i]
```

```
        neighbourData[singleRelativeIndex].add(datavalue)
    else:
                datavalue    =    nodeData[(nodeData.len-1)*(int)(relativePosition_row    !=
2)][(nodeData[0].len-1)*(int)(relativePosition_column != 2)]
      neighbourData[singleRelativeIndex].add(datavalue)


    let dataFile = open($singleRelativeIndex & "data.txt", fmWrite)
    defer: dataFile.close()
    dataFile.writeLine($$neighbourData[singleRelativeIndex])


    nodesParsed += 1
    let parsedFile = open("parsed.txt", fmWrite)
    defer: parsedFile.close()
    parsedFile.writeLine($nodesParsed)
    parsedFile.writeLine($gameStep)
    c.ack( id )


  except CatchableError as e:
    let
     e = getCurrentException()
     msg = getCurrentExceptionMsg()
    echo "Got exception ", repr(e), " with message ", msg
    c.nack( id )
    return
  #do stuff


  if nodesParsed == neighbourNodes:
    padNeighbours()


    var headers: seq[ tuple[name:string, value:string] ]
    headers.add( ("pos-row", $node_position_row) )
    headers.add( ("pos-column", $node_position_column) )
    headers.add( ("type", $1) )
    var nodeDataToSend: seq[int]


    calculateNextStep(layerPointers[0][], layerPointers[gameLayers-1][], true)
    if cellValueChange.row >= 0:
```

```
        layerPointers[cellValueChange.layer][][cellValueChange.row][cellValueChange.column]=
cellValueChange.value
    rearrangeLayers()



    cellValueChange.row = -1


    if gameStep == high(int):
      gameStep = 1
    else:
      gameStep += 1


    drawGrid(layerPointers[0][])
    nodeDataToSend = toSingleDimension(layerPointers[0][])



    nodesParsed = 0


    c.send( "/topic/" & $node_position_row & "." & $node_position_column, $$nodeDataToSend,
"text/plain", headers )
    let parsedFile = open("parsed.txt", fmWrite)
    defer: parsedFile.close()
    parsedFile.writeLine($nodesParsed)
    parsedFile.writeLine($gameStep)


    let selfDataFile  = open("selfData.txt", fmWrite)
    defer: selfDataFile.close()
    selfDataFile.writeLine($$nodeDataToSend)



proc getCPUScores(): seq[int] =
  var cpusFound: int = 0
  var cpuScores = newSeq[int](totalNodes)
  let cpus = open("./cpu_score.txt")
  defer: cpus.close()
  var cpuEntry: string
  while cpus.readLine(cpuEntry):
    var cpuPair = cpuEntry.split("|")
    for i in 0 ..< totalNodes:
```

```
      if cpuPair[0] == nodeCpus[i]:
        cpuScores[i] = parseInt(cpuPair[1])
        cpusFound += 1
    if cpusFound == totalNodes:
      break
  result = cpuScores


proc saveSpecs()=


  var cpu:string = getCpuName().replace("(TM)", "").replace("(R)", "").replace("CPU ", "")
        var        allowedLocalMem        =        getFreeMemory().float*allowedMemoryUsage
 #float(system.getFreeMem())
  nodeCpus[0] = cpu
  nodeMemory[0] = floor(allowedLocalMem)


proc getNeighbourIndexes(nodeIndex: int): seq[int] =


  var nodePosition = nodePositions_new[nodeIndex]
  var neighbours = newSeq[int]()
  for x in max(0, nodePosition.row - 1) .. min(nodePosition.row + 1, activeNodesPerRow.len-
1):
      for y in max(0, nodePosition.column - 1) .. min(nodePosition.column + 1,
activeNodesPerRow[x]-1):
      if (x,y) != nodePosition:
        neighbours.add(nodePositions_new.find((x,y)))


  result = neighbours


proc getMaxNeighbourIndex(nodeIndex: int, requireUnique: bool = false, equalMaxNodeIndex:
int = -1): int =
  #var result = 0
  var maxNeighbourData = 0
  var maxNeighbourIndex = -1
  var node = nodePositions_new[nodeIndex]
  var neighbours = getNeighbourIndexes(nodeIndex)
  var areMaxNeighbours = newSeq[int]()


  for i in 0 ..< neighbours.len:
    var neighbourIndex = neighbours[i]
```

```
    var neighbourData = nodeDataLength[neighbourIndex].rows
     #if (neighbourData == maxNeighbourData) and (maxNeighbourIndex != equalMaxNodeIndex):
continue
    if neighbourData == maxNeighbourData:
      areMaxNeighbours.add(neighbourIndex)

    if neighbourData > maxNeighbourData:
      maxNeighbourData = neighbourData
      maxNeighbourIndex = neighbourIndex
      areMaxNeighbours.setLen(0)
      areMaxNeighbours.add(neighbourIndex)

  result = areMaxNeighbours.findFirstIt(it == equalMaxNodeIndex)

  if equalMaxNodeIndex < 0:
    result = areMaxNeighbours[0]

  if requireUnique and areMaxNeighbours.len > 1:
    result = -1

  #result

proc getNeighboursToCheck(nodeIndex: int, requireUnique: bool = false): seq[int] =
  #var forsterNodeData = nodePositions_new[nodeIndex]
  var neighboursToCheck = newSeq[int]()
  var neighbours = getNeighbourIndexes(nodeIndex)

  for i in 0 ..< neighbours.len:
     var maxNeighbourOfNeighbourIndex = getMaxNeighbourIndex(neighbours[i], requireUnique,
nodeIndex) #changed from i to neighbours[i]
      if maxNeighbourOfNeighbourIndex == nodeIndex: neighboursToCheck.add(neighbours[i])
#changed from i to neighbours[i]

  result = neighboursToCheck

proc subscribeToBlock( c: StompClient, prefetchHeaders: seq[ tuple[name:string,
value:string] ]) =
  neighbourNodes = 0
```

```
    for x in max(0, node_position_row - 1) .. min(node_position_row + 1,
activeNodesPerRow.len - 1):
          for y in max(0, node_position_column - 1) .. min(node_position_column + 1,
activeNodesPerRow[x] - 1):
          if (x != node_position_row) or (y != node_position_column):
            neighbourNodes += 1
                c.subscribe( "/topic/" & $x & "." & $y, "client-individual", "",
prefetchHeaders )


proc createPartitionTable() =


  for i in 0 ..< activeNodesPerRow.len:
    var partitionL = newSeq[Partition]()
    for j in 0 ..< activeNodesPerRow[i]:
      partitionL.add(Partition())


    partitionTable.add(partitionL)


  for i in 0 ..< activeNodesPerRow.len:
    for j in 0 ..< activeNodesPerRow[i]:
      var nodeIdxRow = 0
      var nodeIdxCol = 0
      while nodeIdxRow < i:
          var index = nodePositions_new.findFirstIt(it.row == nodeIdxRow and it.column ==
nodeIdxCol)
        partitionTable[i][j].rowStart += nodeDataLength[index].rows
        partitionTable[i][j].rowEnd += nodeDataLength[index].rows
        nodeIdxRow += 1


      while nodeIdxCol < j:
          var index = nodePositions_new.findFirstIt(it.row == nodeIdxRow and it.column ==
nodeIdxCol)
        partitionTable[i][j].columnStart += nodeDataLength[index].columns
        partitionTable[i][j].columnEnd += nodeDataLength[index].columns
        nodeIdxCol += 1


       var selfIdx = nodePositions_new.findFirstIt(it.row == nodeIdxRow and it.column ==
nodeIdxCol)
      partitionTable[i][j].rowEnd += nodeDataLength[selfIdx].rows - 1
```

```
      partitionTable[i][j].columnEnd += nodeDataLength[selfIdx].columns - 1



proc memArrange() =
  #var memNeeded = sizeof(int)*((2*totalCellNum)+1) #$$
  #var hasSufficientLocalMem = int(nodeMemory[0]*allowedMemoryUsage) > memNeeded #$$


  nodePositions_new = newSeq[ tuple[row:int, column:int]]()
  var nodeWeightedMemory = newSeq[float]()
  #nodeWeightedMemory.add(0.0) #$$
  #nodePositions_new.add((0,0)) #$$

  var nodeCpuScores = getCPUScores()


  var memoryUsed = 0.0 #memorySorted[0].memory #$$
  var scoreTotal = 0 #nodeCpuScores[0] #$$
  var benchTotal = 0.0 #nodeCpuBench[0] #$$

  insufficientMemory = true #not hasSufficientLocalMem #$$



  while (nodesNeeded < totalNodes) and (insufficientMemory):

    memoryUsed += memorySorted[nodesNeeded].memory

    scoreTotal += nodeCpuScores[memorySorted[nodesNeeded].nodeIndex]
    benchTotal += nodeCpuBench[memorySorted[nodesNeeded].nodeIndex]

    nodeDataLength.add((0,0))
    nodeWeightedPerformance.add(0.0)
    nodePositions_new.add((0,0))
    nodeWeightedMemory.add(0.0)
```

```
nodesNeeded += 1

var activeNodeRows = (int)sqrt((float)nodesNeeded)
var edgeNodesTotal = nodesNeeded - (activeNodeRows*activeNodeRows)
#var nodesNeededF = (float)nodesNeeded

#shallow mem check
var memoryNeeded_s = sizeof(int)*2*totalCellNum
var neighbourFences = (int)(nodesNeeded > 3)*(4*activeNodeRows*(activeNodeRows-1))
neighbourFences += edgeNodesTotal*2
var fenceLength = floorDiv(matrixLength, activeNodeRows)
memoryNeeded_s += sizeof(int)*fenceLength*(int)neighbourFences



if memoryNeeded_s > int(memoryUsed): continue

#setting the values for weighted performance and memory
for i in 0 ..< nodesNeeded:
  var index = memorySorted[i].nodeIndex
  var scoreRatio = nodeCpuScores[index]/scoreTotal
  var benchRatio = nodeCpuBench[index]/benchTotal
  nodeWeightedPerformance[i] = (0.5*scoreRatio) + (0.5*benchRatio)
  nodeWeightedMemory[i] = memorySorted[i].memory/memoryUsed



# Arranging the nodes on the "grid"

activeNodesPerRow = activeNodesPerRow.mapIt(0)
if activeNodesPerRow.len < activeNodeRows:
   activeNodesPerRow.add(0)

var edgeNodeColumns = floorDiv(edgeNodesTotal, activeNodeRows)  #was (int) cast
var strandedNodes = edgeNodesTotal mod activeNodeRows

for i in 0 ..< activeNodeRows:
  activeNodesPerRow[i] = activeNodeRows + edgeNodeColumns
  if i < strandedNodes: activeNodesPerRow[i] += 1
```

```
    # Setting each of the nodes' initial size to an equal value, in a way that all data is
included, while also making rearranging it an easy task
    var sliceSquareCellnum = floorDiv(totalCellNum, nodesNeeded)
    var sliceSquareLengthF: float = sqrt((float)sliceSquareCellnum)

    var strandedCells:int = 0
    var migrateCellRows:int = 0
    var sliceSquareLength:int = (int)sliceSquareLengthF
    var orphanCells = totalCellNum - (sliceSquareCellnum*nodesNeeded)
    migrateCellRows = floorDiv(orphanCells, sliceSquareLength)
    strandedCells = orphanCells mod sliceSquareLength

    while (sliceSquareLengthF != (float)floor(sliceSquareLengthF)) or (strandedCells > 0):

      sliceSquareCellnum -= 1
      sliceSquareLengthF = sqrt((float)sliceSquareCellnum)
      sliceSquareLength = (int)sliceSquareLengthF
      orphanCells = totalCellNum - (sliceSquareCellnum*nodesNeeded)
      migrateCellRows = floorDiv(orphanCells, sliceSquareLength)
      strandedCells = orphanCells mod sliceSquareLength


    nodeDataLength = nodeDataLength.mapIt((sliceSquareLength, sliceSquareLength))

    var nodeIdx = 0
    while migrateCellRows > 0:
       nodeDataLength[nodeIdx mod nodesNeeded].rows += 1 #if the number of rows to migrate
is greater than the number of nodes needed, loop through the nodes again.
      migrateCellRows -= 1
      nodeIdx += 1


    #arranging the nodes acording to their available memory
    for i in 0 ..< activeNodeRows:
      nodeIdx = i
      for j in 0 ..< activeNodesPerRow[i]:
```

```
        nodePositions_new[nodeIdx] = (i,j)
        if (j>activeNodeRows-1):
            nodeIdx += activeNodesPerRow.countIt(it > j)    #taking into account the extra
"node columns"
        else:
          nodeIdx += activeNodeRows



    #recording the available memory left on each node
             memoryLeftSortedBySpec      =      newSeq[tuple[memory:float,      specs:float,
nodeIndex:int]](nodesNeeded)
    var memoryLeft = newSeq[float](nodesNeeded)
    for i in 0 ..< nodesNeeded:
      var nodeMemNeeded = nodeDataLength[i].rows*nodeDataLength[i].columns*2



      nodeMemNeeded += nodeDataLength[i].rows*2 + nodeDataLength[i].columns*2 + 5
      if nodesNeeded > 1:
        var maxNeighbourIdx = getMaxNeighbourIndex(i)
                                                                  nodeMemNeeded                +=
nodeDataLength[maxNeighbourIdx].rows*nodeDataLength[maxNeighbourIdx].columns


      nodeMemNeeded *= sizeof(int)



      memoryLeft[i] = memorySorted[i].memory - float(nodeMemNeeded)
      memoryLeftSortedBySpec[i] = (memoryLeft[i], nodeWeightedPerformance[i], i)



    #sorting the available memory table by the processing power of the respective node
     memoryLeftSortedBySpec.sort do (x, y: tuple[memory:float, specs:float, nodeIndex:int])
-> int:
      result = cmp(y.specs, x.specs)



     var  singleRowMemNeededNeighbour  =  float(nodeDataLength[0].columns*sizeof(int))
#every node has the same amount of columns at this point
    var singleRowMemNeeded = 2*(singleRowMemNeededNeighbour + (float)sizeof(int))
    var maxAllowedRows = int(nodeAllowedSize/(nodeDataLength[0].columns*sizeof(int)))
```

```
    var nodeRowCount = newSeq[tuple[rows:int, index:int]]()



    for i in 0 ..< nodesNeeded:
      var j = 0
      var indexOf_i = memoryLeftSortedBySpec[i].nodeIndex
      var neighboursToCheck_i = getNeighboursToCheck(indexOf_i, true)
                        if      (memoryLeftSortedBySpec[i].memory      <=      0.0)      or
neighboursToCheck_i.anyIt(memoryLeft[it]  <=  0.0)  or  (nodeDataLength[indexOf_i].rows  >
maxAllowedRows):
        while j < nodesNeeded:
          var indexOf_j = memoryLeftSortedBySpec[j].nodeIndex
          var neighboursToCheck_j = getNeighboursToCheck(indexOf_j)
             if (i == j) or (memoryLeftSortedBySpec[j].memory <= singleRowMemNeeded) or
neighboursToCheck_j.anyIt(memoryLeft[it]      <=      singleRowMemNeededNeighbour)      or
(nodeDataLength[indexOf_j].rows    >=    (2*nodeDataLength[indexOf_i].rows    +    1))    or
(nodeDataLength[indexOf_j].rows+1 > maxAllowedRows):
            j += 1
            continue


          nodeDataLength[indexOf_i].rows -= 1
          nodeDataLength[indexOf_j].rows += 1
          memoryLeft[indexOf_i] += singleRowMemNeeded
          memoryLeft[indexOf_j] -= singleRowMemNeeded
          memoryLeftSortedBySpec[i].memory += singleRowMemNeeded
          memoryLeftSortedBySpec[j].memory -= singleRowMemNeeded
          for nodeToUpdateIndex in items(neighboursToCheck_i):
              var  nodeAltSortedIndex  =  memoryLeftSortedBySpec.findFirstIt(it.nodeIndex  ==
nodeToUpdateIndex)
            memoryLeft[nodeToUpdateIndex] += singleRowMemNeededNeighbour
                                    memoryLeftSortedBySpec[nodeAltSortedIndex].memory     +=
singleRowMemNeededNeighbour


          for nodeToUpdateIndex in items(neighboursToCheck_j):
              var  nodeAltSortedIndex  =  memoryLeftSortedBySpec.findFirstIt(it.nodeIndex  ==
nodeToUpdateIndex)
            memoryLeft[nodeToUpdateIndex] -= singleRowMemNeededNeighbour
                                    memoryLeftSortedBySpec[nodeAltSortedIndex].memory     -=
singleRowMemNeededNeighbour
```

```
    if memoryLeft.anyIt(it <= 0.0): continue
    insufficientMemory = false


proc simpleArrange() =

  activeNodesPerRow = newSeq[int]()
  nodeDataLength = newSeq[tuple[rows:int, columns:int]](nodesNeeded)
  nodePositions_new = newSeq[tuple[row:int, column:int]](nodesNeeded)
  var gridLength = (int)sqrt((float)nodesNeeded)

  for i in 0 ..< gridLength:
    activeNodesPerRow.add(gridLength)

  var nodeDataSimp= int(matrixLength/gridLength)

  for i in 0 ..< nodesNeeded:
    nodeDataLength[i] = (nodeDataSimp, nodeDataSimp)




  var nodeIdx = 0
  for i in 0 ..< activeNodesPerRow.len:
    for j in 0 ..< activeNodesPerRow[i]:
      nodePositions_new[nodeIdx] = (i, j)
      nodeIdx += 1



proc main() =

  initialize()
  let socket = newSocket()

  let uri = "stomp://" & username & ":" & password & "@" & address & "/"

  var client = newStompClient( socket, uri )
```

```
  client.connected_callback = connected


  client.connect


  var prefetch: seq[ tuple[name:string, value:string] ]
  prefetch.add( ("prefetch-count", $1) )


  if wasInterrupted:


    var headers: seq[ tuple[name:string, value:string] ]
    headers.add( ("pos-row", $node_position_row) )
    headers.add( ("pos-column", $node_position_column) )


    subscribeToBlock(client, prefetch)
    client.message_callback = dataMessageLoop


    if nodesParsed == neighbourNodes:
      var nodeDataToSend = toSingleDimension(layerPointers[0][])
         client.send(  "/topic/"  &  $node_position_row  &  "."  &  $node_position_column,
$$nodeDataToSend, "text/plain", headers )
      nodesParsed = 0


    client.wait_for_messages()
    return


  client.subscribe( "/topic/nodeinit", "client-individual", "", prefetch )


  client.message_callback = initMessage


  client.wait_for_messages(false)
   #matrixLength=  readLineFromStdin("Enter  the  size  of  the  square  matrix  NxN.  N=
").parseInt()
  #totalCellNum = matrixLength*matrixLength
  #allowedMemoryUsage = readLineFromStdin("Enter the percentage of free memory to be used:
 ").parseFloat()
  #allowedMemoryUsage /= 100
  saveSpecs()
```

```
GC_fullcollect()




if totalNodes < 2:
  echo "No nodes available for the distribution of data. Terminating"
  return

prepareForMessages()

try:

  client.message_callback = specMessage
  client.subscribe( "/topic/specshare", "client-individual", "", prefetch )
  client.send( "/topic/specrequest", $allowedMemoryUsage )
  while nodesParsed < totalNodes:
    client.wait_for_messages(false)
except CatchableError:
  let
    e = getCurrentException()
    msg = getCurrentExceptionMsg()
  echo "Got exception ", repr(e), " with message ", msg



nodesParsed = 0
client.unsubscribe("/topic/specshare")

selfIndex = memorySorted.findFirstIt(it.nodeIndex == 0)
for i in 0 ..< totalNodes:
  memorySorted[i] = (nodeMemory[i], i)

memorySorted.sort do (x, y: tuple[memory:float, nodeIndex:int]) -> int:
  result = cmp(y.memory, x.memory)
  if x.nodeIndex == 0:
    result = -1

if not isSimpleDist:
```

```
    nodesNeeded = 0  #1 #$$
    memArrange()
    if nodesNeeded == totalNodes and (insufficientMemory):
      var pause = readLineFromStdin("Not enough resources. Terminating.")
      return
  else:
    simpleArrange()
    insufficientMemory = false



  createPartitionTable()


  echo "Initializing grid with " & $nodesNeeded & " nodes."
   #NETWORK STEP 6: sending to each node their new position on the grid and their data
lengths
  for i in nodesNeeded ..< totalNodes:
    var k = memorySorted[i].nodeIndex
    var currentPos = nodePositions[k]
    var dimensions = newSeq[int]()
    dimensions.add(0)
    dimensions.add(0)
    var headers: seq[ tuple[name:string, value:string] ]
    headers.add( ("isActive", $0) )
    client.send( "/topic/info" & $currentPos.row & "." & $currentPos.column, $$dimensions ,
"text/plain", headers )  # $$ convert to json


  var matrix = matrixData.distribute(matrixLength, false)
  for i in 0 ..< nodesNeeded:
    var k = memorySorted[i].nodeIndex
    var currentPos = nodePositions[k]
    var newPos = nodePositions_new[i]
    if k != selfIndex:
      var dimensions = newSeq[int]()
      dimensions.add(nodeDataLength[i].rows)
      dimensions.add(nodeDataLength[i].columns)
      var headers: seq[ tuple[name:string, value:string] ]
      headers.add( ("pos-row", $newPos.row) )
      headers.add( ("pos-column", $newPos.column) )
```

```
      headers.add( ("layers", $gameLayers) )
      headers.add( ("isActive", $1) )
      headers.add( ("activeNodes", $$activeNodesPerRow) )
      headers.add( ("dimensions", $$dimensions) )
      var seqToSend: seq[int]
      if not isRandom:
        var rowStart = nodeDataLength[i].rows*newPos.row
        var rowEnd = rowStart + nodeDataLength[i].rows  - 1
        var columnStart = nodeDataLength[i].columns*newPos.column
        var columnEnd = columnStart + nodeDataLength[i].columns - 1
        var dataToSend= getSeqSlice(matrix, rowStart, rowEnd, columnStart, columnEnd)
        seqToSend = toSingleDimension(dataToSend)
        #echo $(rowStart, rowEnd, columnStart, columnEnd)
        #echo $dataToSend
        #var asd = readLineFromStdin("asd ")
      client.send( "/topic/info" & $currentPos.row & "." & $currentPos.column, $$seqToSend,
"text/plain", headers )  # $$ convert to json
    nodePositions[k] = nodePositions_new[i]



        (node_position_row,       node_position_column)       =       (nodePositions[0].row,
nodePositions[0].column)
  nodeDataLengthSelf = newSeq[int]()
  nodeDataLengthSelf.add(nodeDataLength[0].rows)
  nodeDataLengthSelf.add(nodeDataLength[0].columns)
  layerPointers = newSeq[ptr seq[seq[int]]](gameLayers)
  gameField = newSeq[seq[seq[int]]](gameLayers)
  for i in 0 ..< gameLayers:
    gameField[i] = newSeqWith(nodeDataLengthSelf[0], newSeq[int](nodeDataLengthSelf[1]))
    layerPointers[i] = addr gameField[i]

  localRowLength = nodeDataLengthSelf[0]
  localColumnLength = nodeDataLengthSelf[1]



  let positions = open("positions.txt", fmWrite)
  defer: positions.close()
  positions.writeLine($node_position_row)
```

```
  positions.writeLine($node_position_column)
  positions.writeLine($$nodeDataLengthSelf)
  positions.writeLine($$activeNodesPerRow)

  var headers: seq[ tuple[name:string, value:string] ]
  headers.add(("row-len", $nodeDataLengthSelf[0]))
  headers.add(("activeNodes", $$activeNodesPerRow))
  client.send( "/topic/gatewayinfo", $$partitionTable, "text/plain", headers )

  subscribeToBlock(client, prefetch)



  if isRandom:
    fillGrid(layerPointers[0][], nodeDataLengthSelf[0], nodeDataLengthSelf[1])
  else:
        layerPointers[0][]   =   getSeqSlice(matrix,   0,   nodeDataLengthSelf[0]-1,   0,
nodeDataLengthSelf[1]-1)

  gameStep += 1
  drawGrid(layerPointers[0][])



  sleep(1000)
  var nodeDataToSend = toSingleDimension(layerPointers[0][])
  headers.setLen(0)
  headers.add( ("pos-row", $node_position_row) )
  headers.add( ("pos-column", $node_position_column) )
  headers.add( ("type", $1) )
    client.send(   "/topic/"   &   $node_position_row   &   "."   &   $node_position_column,
$$nodeDataToSend, "text/plain", headers )

  if nodesNeeded == 1:
    while true:
      calculateNextStep(layerPointers[0][], layerPointers[gameLayers-1][], false)
      rearrangeLayers()

      if gameStep == high(int):
        gameStep = 1
```

```
      else:
        gameStep += 1


      drawGrid(layerPointers[0][])


      nodeDataToSend = toSingleDimension(layerPointers[0][])
         client.send(  "/topic/"  &  $node_position_row  &  "."  &  $node_position_column,
$$nodeDataToSend, "text/plain", headers )


    system.quit()


  client.message_callback = dataMessageLoop
  nodeDataToSend.setLen(0)
  GC_fullcollect()



  nodesParsed = 0
  client.wait_for_messages()


main()
```

gol_services_2.nim (slave nodes)

```
#!nimrod


import
   std/[net,json],
   stomp,
   strutils,
   sequtils,
   std/[os,random],
   marshal,
   rdstdin,
   sysinfo,
   std/monotimes,
```

```
   macros

when NimMajor < 2:
  import system/io

from times import inMilliseconds

var username, password, address: string
var node_position_row, node_position_column: int
var nodes_length_row, nodes_length_column: int

var nodeDataLengthSelf: seq[int]

var neighbourNodes: int = 0
var totalNodes: int = 1
var nodesParsed: int = 0
var activeNodesPerRow: seq[int]

var wasInterrupted: bool = false

var neighbourData: seq[seq[int]]
var gameField: seq[seq[seq[int]]]

var initData: seq[seq[int]]
var isRandom: bool = true
var gameStep: int
var gameLayers: int

var cellValueChange: tuple[layer, row, column, value:int] = (-1, -1, -1, -1)

var layerPointers: seq[ptr seq[seq[int]]]

#--------------------------------------------------------------------------BENCHMARK-----
-------------------------------------------------------------------------------------------
-----
const cLOOPS {.intdefine.} = 1225

# avoids some "bit-twiddling" for better speed...
```

```
const cBITMSK = [ 1'u8, 2, 4, 8, 16, 32, 64, 128 ]

macro unrollLoops(ca, sz, strtndx, bp: untyped) =
  let cmpstsalmtid = newIdentNode("cmpstsalmt")
  let szbitsid = newIdentNode("szbits")
  let strtndx0id = newIdentNode("strtndx0")
  let strt0id = newIdentNode("strt0")
  let strt7id = newIdentNode("strt7")
  let endalmtid = newIdentNode("endalmt")
  let bpintid = newIdentNode("bpint")
  let cullaid = newIdentNode("culla")
  result = quote do:
    let `szbitsid` = `sz` shl 3
    let `cmpstsalmtid` = `ca` + `sz`
    let `bpintid` = `bp`.int
    let `strtndx0id` = `strtndx`
    let `strt0id` = `strtndx0id` shr 3
  for i in 1 .. 7:
    let strtndxido = newIdentNode("strtndx" & $(i - 1))
    let strtndxidn = newIdentNode("strtndx" & $i)
    let strtid = newIdentNode("strt" & $i)
    result.add quote do:
      let `strtndxidn` = `strtndxido` + `bp`
      let `strtid` = (`strtndxidn` shr 3) - `strt0id`
  let csstmnt = quote do:
    case (((`bpintid` and 0x6) shl 2) + (`strtndx0id` and 7)).uint8
    of 0'u8: break
  csstmnt.del 1 # delete last dummy "of"
  for n in 0'u8 .. 0x3F'u8: # actually used cases...
    let pn = (n shr 2) or 1'u8
    let cn = n and 7'u8
    let mod0id = newLit(cn)
    let cptr0id = "cptr0".newIdentNode
    let loopstmnts = nnkStmtList.newTree()
    for i in 0'u8 .. 7'u8:
      let mskid = newLit(1'u8 shl ((cn + pn * i.uint8) and 7).int)
      let cptrid = ("cptr" & $i).newIdentNode
```

```
      let strtid = ("strt" & $i).newIdentNode
      if i == 0'u8:
        loopstmnts.add quote do:
          let `cptrid` = cast[ptr uint8](`cullaid`)
      else:
        loopstmnts.add quote do:
          let `cptrid` = cast[ptr uint8](`cullaid` + `strtid`)
      loopstmnts.add quote do:
        `cptrid`[] = `cptrid`[] or `mskid`
    loopstmnts.add quote do:
      `cullaid` += `bpintid`
    let ofbrstmnts = quote do:
      while `cullaid` < `endalmtid`:
        `loopstmnts`
      `cullaid` = ((`cullaid` - `ca`) shl 3) or `mod0id`.int
      while `cullaid` < `szbitsid`:
        let `cptr0id` = cast[ptr uint8](`ca` + (`cullaid` shr 3))
        `cptr0id`[] = `cptr0id`[] or cBITMSK[`cullaid` and 7]
        `cullaid` += `bpintid`
    csstmnt.add nnkOfBranch.newTree(
      newLit(n),
      ofbrstmnts
    )
  for n in 0x40'u8 .. 255'u8: # fill in defaults for remaining possibilities
    csstmnt.add nnkOfBranch.newTree(
      newLit(n),
      nnkStmtList.newTree(
        nnkBreakStmt.newTree(
          newEmptyNode()
        )
      )
    )
  result.add quote do:
    let `endalmtid` = `cmpstsalmtid` - `strt7id`
    var `cullaid` = `ca` + `strt0id`
    `csstmnt`
#  echo csstmnt[9].astGenRepr # see AST for a given case
```

```
#  echo csstmnt[9].toStrLit # see code for a given case
#  echo result.toStrLit # see entire produced code at compile time


proc benchSoE(): iterator(): int {.closure.} =
  var cmpsts = newSeq[byte](16384)
  let cmpstsa = cast[int](cmpsts[0].addr)
  for _ in 0 ..< cLOOPS:
    for i in 0 .. 254: # cull to square root of limit
      if (cmpsts[i shr 3] and cBITMSK[i and 7]) == 0'u8: # if prime -> cull its composites
        let bp = i +% i +% 3
        let swi = (i +% i) *% (i +% 3) +% 3
        unrollLoops(cmpstsa, 16384, swi, bp)
  return iterator(): int {.closure.} =
    yield 2 # the only even prime
    for i in 0 .. 131071: # separate iteration over results
      if (cmpsts[i shr 3] and cBITMSK[i and 7]) == 0'u8:
        yield i +% i +% 3
#------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------


proc cleanScreen*() =
     write(stdout,"\e[H\e[J")


proc padNeighbours() =
  for i in [0,2,6,8]:
    if neighbourData[i].len == 0: neighbourData[i].add(0)


  while neighbourData[3].len < nodeDataLengthSelf[0]:
    neighbourData[3].add(0)
  while neighbourData[5].len < nodeDataLengthSelf[0]:
    neighbourData[5].add(0)


  while neighbourData[1].len < nodeDataLengthSelf[1]:
    neighbourData[1].add(0)
  while neighbourData[7].len < nodeDataLengthSelf[1]:
    neighbourData[7].add(0)
```

```
proc fillGrid(gameField: var seq[seq[int]], rows, columns: int) =
    var seed=0
    var value=0
    for i in 0 ..< rows:
        for j in 0 ..< columns:
          seed = rand(100)
          value = (int)seed>86
          gameField[i][j] = value

proc drawGrid(gameField: seq[seq[int]]) =
  cleanScreen()
  for i in 0 ..< gameField.len: #nodeDataLengthSelf[0]:
     for j in 0 ..< gameField[i].len: #nodeDataLengthSelf[1]:
         var cellState = if (bool)gameField[i][j]: "0" else: " "
         stdout.write cellState

     echo " "
  echo " "
  echo "Step " & $gameStep

proc rearrangeLayers() =
  var tempPtr = layerPointers[gameLayers-1]
  for i in countdown(layerPointers.len-1,1):
    layerPointers[i] = layerPointers[i-1]
  layerPointers[0] = tempPtr

proc calculateNeighbours(i,j: int, gameField: seq[seq[int]]): int =
  var neighbours = 0;
  for x in max(0, i - 1) .. min(i + 1, nodeDataLengthSelf[0]-1):
    for y in max(0, j - 1) .. min(j + 1, nodeDataLengthSelf[1]-1):
      if x != i or y != j: neighbours += gameField[x][y]

  result= neighbours

proc calculateBorders(currentStep, nextStep: var seq[seq[int]]) =
  for i in 0 ..< nodeDataLengthSelf[0]:
```

```
    var lastRowElem = nodeDataLengthSelf[1]-1
    var aliveNeighboursL = calculateNeighbours(i, 0, currentStep)
    var aliveNeighboursR = calculateNeighbours(i, lastRowElem, currentStep)

    if i == 0:
      aliveNeighboursL += neighbourData[0][0]
      aliveNeighboursL += neighbourData[1][0]
      aliveNeighboursL += neighbourData[1][1]
      aliveNeighboursR += neighbourData[2][0]
      aliveNeighboursR += neighbourData[1][lastRowElem-1]
      aliveNeighboursR += neighbourData[1][lastRowElem]

    if i == nodeDataLengthSelf[0]-1:
      aliveNeighboursL += neighbourData[6][0]
      aliveNeighboursL += neighbourData[7][0]
      aliveNeighboursL += neighbourData[7][1]
      aliveNeighboursR += neighbourData[8][0]
      aliveNeighboursR += neighbourData[7][lastRowElem-1]
      aliveNeighboursR += neighbourData[7][lastRowElem]
    else:
      aliveNeighboursL += neighbourData[3][i+1]
      aliveNeighboursR += neighbourData[5][i+1]

    aliveNeighboursL += neighbourData[3][i]
    aliveNeighboursR += neighbourData[5][i]

        nextStep[i][0]   =   (int)((aliveNeighboursL   or   currentStep[i][0])   ==   3)
#(int)((aliveNeighboursL == 2) or (aliveNeighboursL == 3))
     nextStep[i][lastRowElem] = (int)((aliveNeighboursR or currentStep[i][lastRowElem]) ==
3) #(int)((aliveNeighboursR == 2) or (aliveNeighboursR == 3))

  for j in 0 ..< nodeDataLengthSelf[1]-1:
    var lastColumnElem = nodeDataLengthSelf[0]-1
    var aliveNeighboursU = calculateNeighbours(0, j, currentStep)
    var aliveNeighboursD = calculateNeighbours(lastColumnElem, j, currentStep)
    aliveNeighboursU += neighbourData[1][j] + neighbourData[1][j+1]
    aliveNeighboursD += neighbourData[7][j] + neighbourData[7][j+1]
    nextStep[0][j] = (int)((aliveNeighboursU or currentStep[0][j]) == 3)
```

```
                nextStep[lastColumnElem][j]       =       (int)((aliveNeighboursD       or
currentStep[lastColumnElem][j]) == 3)


proc calculateNextStep(currentStep, nextStep: var seq[seq[int]]) =


  for i in 1 ..< nodeDataLengthSelf[0]-1:
    for j in 1 ..< nodeDataLengthSelf[1]-1:
      var aliveNeighbours = calculateNeighbours(i, j, currentStep)


      nextStep[i][j] = (int)((aliveNeighbours or currentStep[i][j]) == 3)


  calculateBorders(currentStep,nextStep)


proc toSingleDimension(seqtoedit: seq[seq[int]]): seq[int] =
  #var result = newSeq[int]()
  for i in 0 ..< seqtoedit.len:
    for j in 0 ..< seqtoedit[i].len:
      result.add(seqtoedit[i][j])
  result


proc initialize() =
  randomize()


  neighbourData = newSeqWith(9, newSeq[int](1))


  let creds = open("./credentials.txt")
  defer: creds.close()
  node_position_row = parseInt(creds.readLine())
  node_position_column = parseInt(creds.readLine())
  nodes_length_row = parseInt(creds.readLine())
  nodes_length_column = parseInt(creds.readLine())
  address = creds.readLine()
  username = creds.readLine()
  password = creds.readLine()
  totalNodes = nodes_length_row*nodes_length_column
  nodesParsed = 0
```

```
  if fileExists("./positions.txt"):

    var continueStep = readLineFromStdin("The daemon was terminated unexpectedly. Continue?
(Y/n): ")
    if continueStep.toUpperAscii != "Y": return

    wasInterrupted = true
    let positions = open("./positions.txt")
    defer: positions.close()
    node_position_row = parseInt(positions.readLine())
    node_position_column = parseInt(positions.readLine())
    nodeDataLengthSelf = to[seq[int]](positions.readLine())
    activeNodesPerRow = to[seq[int]](positions.readLine())
    if fileExists("./selfData.txt"):
      let selfDataFile  = open("selfData.txt")
      defer: selfDataFile.close()
      var selfData = to[seq[int]](selfDataFile.readLine())
      gameField = newSeq[seq[seq[int]]](gameLayers)

      var nodeData = selfData.distribute(nodeDataLengthSelf[0], false)
      layerPointers = newSeq[ptr seq[seq[int]]](gameLayers)
      for i in 0 ..< gameLayers:
        gameField[i] = nodeData
        layerPointers[i] = addr gameField[0]

  if fileExists("./parsed.txt"):
    let parsedFile = open("parsed.txt")
    defer: parsedFile.close()
    nodesParsed = parseInt(parsedFile.readLine())
    gameStep = parseInt(parsedFile.readLine())
    #todo: read more stuff here(neighbours,self data)

  for i in 0 .. 8:
    if fileExists($i & "data.txt"):
      let dataFile = open($i & "data.txt")
      defer: dataFile.close()
      var data = to[seq[int]](dataFile.readLine())
```

```
      neighbourData[i] = data




# Announce when we're connected.
proc connected( c: StompClient, r: StompResponse ) =
   echo "Connected to a ", c["server"], " server."



#NETWORK STEP 3: send the specs that were requested
proc specMessage( c: StompClient, r: StompResponse ) =
  let id = r[ "ack" ]
  var allowedMemoryUsage: float

  try:
     allowedMemoryUsage = parseFloat(r.payload)
  except CatchableError as e:
     echo "Couldn't parse! ", r.payload
     c.nack( id )

  GC_fullcollect()
  var allowedLocalMem = float(getFreeMemory())
  let allowedNodeMem = (int)(allowedLocalMem*allowedMemoryUsage)
  #echo $(allowedLocalMem, allowedNodeMem)
  #var ss = readLineFromStdin("ses")
  var cpu = getCpuName().replace("(TM)", "").replace("(R)", "").replace("CPU ", "")

  let strt = getMonotime()
  let answr = benchSoE()
  var cnt = 0; for _ in answr(): cnt += 1
  var elpsd = float((getMonotime() - strt).inMilliseconds)
  elpsd = 100/elpsd



  var headers: seq[ tuple[name:string, value:string] ]
  headers.add( ("mem", $allowedNodeMem) )
```

```
  headers.add( ("bench", $elpsd) )
  headers.add( ("pos-row", $node_position_row) )
  headers.add( ("pos-column", $node_position_column) )
  c.send( "/topic/specshare", cpu, "text/plain", headers )
  c.ack( id )

#NETWORK STEP 7: process the new position and data
proc infoMessage( c: StompClient, r: StompResponse ) =
  let id = r[ "ack" ]
  var isActive = parseInt(r["isActive"])

  if not bool(isActive):
    echo "Not part of the grid. Terminating."
    c.ack( id )
    system.quit()

  var oldRow = node_position_row
  var oldColumn = node_position_column
  echo $r["activeNodes"]

  node_position_row = parseInt(r["pos-row"])
  node_position_column = parseInt(r["pos-column"])
  nodeDataLengthSelf = to[seq[int]](r["dimensions"])

  activeNodesPerRow = to[seq[int]](r["activeNodes"])
  gameLayers = parseInt(r["layers"])

  var data = to[seq[int]](r.payload)
  if data.len > 0:
    isRandom = false
  initData = data.distribute(nodeDataLengthSelf[0],false)



  layerPointers = newSeq[ptr seq[seq[int]]](gameLayers)
  gameField = newSeq[seq[seq[int]]](gameLayers)
  for i in 0 ..< gameLayers:
    gameField[i] = newSeqWith(nodeDataLengthSelf[0], newSeq[int](nodeDataLengthSelf[1]))
```

```
    layerPointers[i] = addr gameField[i]


  c.ack( id )


  let positions = open("positions.txt", fmWrite)
  defer: positions.close()
  positions.writeLine($node_position_row)
  positions.writeLine($node_position_column)
  positions.writeLine(r["dimensions"])
  positions.writeLine(r.payload)



  sleep(100)
  c.unsubscribe( "/topic/info" & $oldRow & "." & $oldColumn)

proc dataMessageLoop( c: StompClient, r: StompResponse ) =


  sleep(1000)
  let id = r[ "ack" ]


  let msgType = parseInt(r["type"])
  if msgType == 3:
    let cellPos_row = parseInt(r["cell-row"])
    let cellPos_col = parseInt(r["cell-col"])
    let layer = parseInt(r["layer"])
    var headers: seq[ tuple[name:string, value:string] ]
    headers.add( ("type", $3) )
       c.send(   "/topic/clientmessage",   $layerPointers[layer][][cellPos_row][cellPos_col],
"text/plain", headers )
    c.ack( id )
    return
  if msgType == 4:
    let cellPos_row = parseInt(r["cell-row"])
    let cellPos_col = parseInt(r["cell-col"])
    let layer = parseInt(r["layer"])
    var valueToSet = parseInt(r.payload)
    cellValueChange = (layer, cellPos_row, cellPos_col, valueToSet)
    c.ack( id )
```

```
    return




  try:
    let rowActual = parseInt(r["pos-row"])
    let columnActual: int = parseInt(r["pos-column"])

    #if rowActual == node_position_row and columnActual == node_position_column: return

    let relativePosition_row = rowActual-node_position_row+1
    let relativePosition_column = columnActual-node_position_column+1
    let singleRelativeIndex = relativePosition_column + 3*relativePosition_row
    var nodeData = to[seq[int]](r.payload).distribute(nodeDataLengthSelf[0], false)
    neighbourData[singleRelativeIndex] = @[]
    var datavalue: int
    if 3 <= singleRelativeIndex and singleRelativeIndex <= 5:

      for i in 0 ..< min(nodeData.len,nodeDataLengthSelf[0]):
        datavalue = nodeData[i][(nodeData[i].len-1)*(int)(singleRelativeIndex==3)]
        neighbourData[singleRelativeIndex].add(datavalue)
    elif singleRelativeIndex mod 3 == 1:
      for i in 0 ..< min(nodeData[0].len,nodeDataLengthSelf[1]):
        datavalue = nodeData[(nodeData.len-1)*(int)(singleRelativeIndex==1)][i]
        neighbourData[singleRelativeIndex].add(datavalue)
    else:
                datavalue    =    nodeData[(nodeData.len-1)*(int)(relativePosition_row    !=
2)][(nodeData[0].len-1)*(int)(relativePosition_column != 2)]
      neighbourData[singleRelativeIndex].add(datavalue)

    let dataFile = open($singleRelativeIndex & "data.txt", fmWrite)
    defer: dataFile.close()
    dataFile.writeLine($$neighbourData[singleRelativeIndex])

    nodesParsed += 1
    let parsedFile = open("parsed.txt", fmWrite)
    defer: parsedFile.close()
    parsedFile.writeLine($nodesParsed)
```

```
    parsedFile.writeLine($gameStep)
    c.ack( id )

  except CatchableError as e:
    let
     e = getCurrentException()
     msg = getCurrentExceptionMsg()
    echo "Got exception ", repr(e), " with message ", msg
    c.nack( id )
    return
  #do stuff

  if nodesParsed == neighbourNodes:
    padNeighbours()

    var headers: seq[ tuple[name:string, value:string] ]
    headers.add( ("pos-row", $node_position_row) )
    headers.add( ("pos-column", $node_position_column) )
    headers.add( ("type", $1) )
    var nodeDataToSend: seq[int]

    calculateNextStep(layerPointers[0][], layerPointers[gameLayers-1][])
    if cellValueChange.row >= 0:
        layerPointers[cellValueChange.layer][][cellValueChange.row][cellValueChange.column]=
cellValueChange.value
    rearrangeLayers()



    cellValueChange.row = -1

    if gameStep == high(int):
      gameStep = 1
    else:
      gameStep += 1

    drawGrid(layerPointers[0][])
    nodeDataToSend = toSingleDimension(layerPointers[0][])
```

```
    nodesParsed = 0


    c.send( "/topic/" & $node_position_row & "." & $node_position_column, $$nodeDataToSend,
"text/plain", headers )
    let parsedFile = open("parsed.txt", fmWrite)
    defer: parsedFile.close()
    parsedFile.writeLine($nodesParsed)
    parsedFile.writeLine($gameStep)


    let selfDataFile  = open("selfData.txt", fmWrite)
    defer: selfDataFile.close()
    selfDataFile.writeLine($$nodeDataToSend)




proc subscribeToBlock( c: StompClient, prefetchHeaders: seq[ tuple[name:string,
value:string] ]) =
  neighbourNodes = 0
    for x in max(0, node_position_row - 1) .. min(node_position_row + 1,
activeNodesPerRow.len - 1):
          for y in max(0, node_position_column - 1) .. min(node_position_column + 1,
activeNodesPerRow[x] - 1):
         if (x != node_position_row) or (y != node_position_column):
           neighbourNodes += 1
                c.subscribe( "/topic/" & $x & "." & $y, "client-individual", "",
prefetchHeaders )



proc main() =
  wasInterrupted = false


  initialize()


  let socket = newSocket()


  let uri = "stomp://" & username & ":" & password & "@" & address & "/"


  var client = newStompClient( socket, uri )
  # Attach callbacks
  client.connected_callback = connected
```

```
  var prefetch: seq[ tuple[name:string, value:string] ]
  prefetch.add( ("prefetch-count", $1) )

  var headers: seq[ tuple[name:string, value:string] ]
  headers.add( ("pos-row", $node_position_row) )
  headers.add( ("pos-column", $node_position_column) )
  headers.add( ("type", $1) )



  if wasInterrupted:
    client.connect
    subscribeToBlock(client, prefetch)
    client.message_callback = dataMessageLoop

    if nodesParsed == neighbourNodes:
      var nodeDataToSend = toSingleDimension(layerPointers[0][])
         client.send(  "/topic/"  &  $node_position_row  &  "."  &  $node_position_column,
$$nodeDataToSend, "text/plain", headers )
      nodesParsed = 0

    client.wait_for_messages()
    return



  try:

    nodesParsed = 0

     #NETWORK STEP 1: subscribe to the spec request that the parent node is going to send
and wait
    client.message_callback = specMessage
    # Open a session with the broker
    client.connect
    client.subscribe( "/topic/specrequest", "client-individual", "", prefetch )
    client.wait_for_messages(false)
```

```
    client.unsubscribe( "/topic/specrequest")


    #NETWORK STEP 5: subscribe and wait for the information
    nodesParsed = 0
    client.message_callback = infoMessage
     client.subscribe( "/topic/info" & $node_position_row & "." & $node_position_column,
"client-individual", "", prefetch )
    client.wait_for_messages(false)
  except CatchableError:
    let
      e = getCurrentException()
      msg = getCurrentExceptionMsg()
    echo "Got exception ", repr(e), " with message ", msg


  subscribeToBlock(client, prefetch)


  if isRandom:
    fillGrid(layerPointers[0][], nodeDataLengthSelf[0], nodeDataLengthSelf[1])
  else:
    layerPointers[0][] = initData


  gameStep += 1
  drawGrid(layerPointers[0][])




  sleep(1000)
  var nodeDataToSend = toSingleDimension(layerPointers[0][])


    client.send(   "/topic/"   &   $node_position_row   &   "."   &   $node_position_column,
$$nodeDataToSend, "text/plain", headers )



  client.message_callback = dataMessageLoop
  nodeDataToSend.setLen(0)
  GC_fullcollect()


  nodesParsed = 0
```

```
  client.wait_for_messages()


main()
```

gol_gateway.nim (gateway)

```
import stomp
import std/net
import rdstdin
import marshal
import strutils

type Partition = object
  rowStart: int
  rowEnd: int
  columnStart: int
  columnEnd: int

var username, password, address: string
var activeNodesPerRow: seq[int]
var rowLength: int
var sub_row, sub_col: int = 0
var isFirstNode = true

var partitionTable: seq[seq[Partition]]

# Announce when we're connected.
proc connected( c: StompClient, r: StompResponse ) =
   echo "Connected to a ", c["server"], " server."

proc initialize() =

  let creds = open("./gateway/credentials.txt")
  defer: creds.close()
  address = creds.readLine()
  username = creds.readLine()
  password = creds.readLine()
```

```
proc infoMessage( c: StompClient, r: StompResponse ) =
  let id = r[ "ack" ]

  try:
    partitionTable = to[seq[seq[Partition]]](r.payload)
    rowLength = parseInt(r["row-len"])
    activeNodesPerRow = to[seq[int]](r["activeNodes"])
    var headers: seq[ tuple[name:string, value:string] ]
    headers.add(("row-len", r["row-len"]))
    c.send( "/topic/clientinfo", r["activeNodes"], "text/plain", headers )
  except CatchableError as e:
     let
       e = getCurrentException()
       msg = getCurrentExceptionMsg()
     echo "Got exception ", repr(e), " with message ", msg
     c.nack( id )
     return


  c.ack( id )

proc nodeDataMessage( c: StompClient, r: StompResponse ) =
  let id = r[ "ack" ]

  try:
     var headers: seq[ tuple[name:string, value:string] ]
     headers.add(("matrix-len", r["matrix-len"]))
     headers.add(("node-num", r["node-num"]))
     headers.add(("layers", r["layers"]))
     headers.add(("mem-use", r["mem-use"]))
     c.send( "/topic/nodeinit", r.payload, "text/plain", headers )
  except CatchableError as e:
     let
       e = getCurrentException()
       msg = getCurrentExceptionMsg()
     echo "Got exception ", repr(e), " with message ", msg
```

```
    c.nack( id )
    return



  c.ack( id )

proc messageLoop( c: StompClient, r: StompResponse ) =
  let id = r[ "ack" ]

  var prefetch: seq[ tuple[name:string, value:string] ]
  prefetch.add( ("prefetch-count", $1) )

  try:
    var messageType = parseInt(r["type"])
    case messageType:
      of 1:
        var headers: seq[ tuple[name:string, value:string] ]
        headers.add(("pos-row", r["pos-row"]))
        headers.add(("type", r["type"]))
        headers.add(("pos-column", r["pos-column"]))
        c.send( "/topic/clientmessage", r.payload, "text/plain", headers )
      of 2:
        if not isFirstNode:
          c.unsubscribe( "/topic/" & $sub_row & "." & $sub_col)


        echo "Subbing to new node"
        var newNode = to[seq[int]](r.payload)
         c.subscribe( "/topic/" & $newNode[0] & "." & $newNode[1], "client-individual", "",
prefetch )
        sub_row = newNode[0]
        sub_col = newNode[1]


        isFirstNode = false
      of 3:
        let cellPos_row = parseInt(r["cell-row"])
        let cellPos_col = parseInt(r["cell-col"])
        for i in 0 ..< activeNodesPerRow.len:
          for j in 0 ..< activeNodesPerRow[i]:
```

```
                    if cellPos_row >= partitionTable[i][j].rowStart and cellPos_row <= partitionTable[i][j].rowEnd and cellPos_col >= partitionTable[i][j].columnStart and cellPos_col <= partitionTable[i][j].columnEnd:
                let cellPos_row_node = cellPos_row - partitionTable[i][j].rowStart
                let cellPos_col_node = cellPos_row - partitionTable[i][j].columnStart
                var headers: seq[ tuple[name:string, value:string] ]
                headers.add(("cell-row", $cellPos_row_node))
                headers.add(("cell-col", $cellPos_col_node))
                headers.add(("layer", r["layer"]))
                headers.add(("type", $3))
                c.send( "/topic/" & $i & "." & $j, $0, "text/plain", headers )
      of 4:
        let cellPos_row = parseInt(r["cell-row"])
        let cellPos_col = parseInt(r["cell-col"])
        for i in 0 ..< activeNodesPerRow.len:
          for j in 0 ..< activeNodesPerRow[i]:
                    if cellPos_row >= partitionTable[i][j].rowStart and cellPos_row <= partitionTable[i][j].rowEnd and cellPos_col >= partitionTable[i][j].columnStart and cellPos_col <= partitionTable[i][j].columnEnd:
                let cellPos_row_node = cellPos_row - partitionTable[i][j].rowStart
                let cellPos_col_node = cellPos_row - partitionTable[i][j].columnStart
                var headers: seq[ tuple[name:string, value:string] ]
                headers.add(("cell-row", $cellPos_row_node))
                headers.add(("cell-col", $cellPos_col_node))
                headers.add(("layer", r["layer"]))
                headers.add(("type", $4))
                c.send( "/topic/" & $i & "." & $j, $r.payload, "text/plain", headers )
      else:
        c.unsubscribe( "/topic/" & $sub_row & "." & $sub_col)


  except CatchableError as e:
    let
      e = getCurrentException()
      msg = getCurrentExceptionMsg()
    echo "Got exception ", repr(e), " with message ", msg
    var h = readLineFromStdin("> ")
    c.nack( id )
    return
```

```
  c.ack( id )

proc main() =

  initialize()
  let socket = newSocket()

  let uri = "stomp://" & username & ":" & password & "@" & address & "/"

  var client = newStompClient( socket, uri )

  # Attach callbacks
  client.connected_callback = connected
  client.connect

  var prefetch: seq[ tuple[name:string, value:string] ]
  prefetch.add( ("prefetch-count", $1) )

  client.message_callback = nodeDataMessage

  client.subscribe( "/topic/gatewaynodeinit", "client-individual", "", prefetch )

  client.wait_for_messages(false)

  client.unsubscribe( "/topic/gatewaynodeinit")

  client.message_callback = infoMessage

  client.subscribe( "/topic/gatewayinfo", "client-individual", "", prefetch )

  client.wait_for_messages(false)

  client.unsubscribe("/topic/gatewayinfo")

  client.subscribe( "/topic/clientrequest", "client-individual", "", prefetch )
```

```
  client.message_callback = messageLoop


  client.wait_for_messages()




main()
```

gol_client.nim (client)

```
# This example shows how to draw the surface of a control.
import std/net
import nigui
import threadpool
import stomp
import strutils
import sequtils
import marshal
import rdstdin
import std/math

const WINDOW_WIDTH = 1200
const WINDOW_HEIGHT = 1000

var gameCanvas: Canvas
var gameStateImage: Image
var control1: Control
var client: StompClient
var comboBoxRow: ComboBox
var comboBoxColumn: ComboBox
var window: Window

var observingNode = false

let socket = newSocket()
```

```
var username, password, address: string
var activeNodesPerRow: seq[int]
var rowLength: int
var sub_row, sub_col: int
var gameState: seq[seq[int]]
var widthOffset, heightOffset: int = 800

proc readCell(layer,cellRow, cellCol: int) =
  var headers: seq[ tuple[name:string, value:string] ]
  headers.add(("cell-row", $cellRow))
  headers.add(("cell-col", $cellCol))
  headers.add(("layer", $layer))
  headers.add(("type", $3))
  client.send( "/topic/clientrequest", $0, "text/plain", headers )

proc writeCell(layer, cellRow, cellCol, value: int) =
  var headers: seq[ tuple[name:string, value:string] ]
  headers.add(("cell-row", $cellRow))
  headers.add(("cell-col", $cellCol))
  headers.add(("layer", $layer))
  headers.add(("type", $4))
  client.send( "/topic/clientrequest", $value, "text/plain", headers )

# Announce when we're connected.
proc connected( c: StompClient, r: StompResponse ) =
   echo "Connected to a ", c["server"], " server."

proc infoMessage( c: StompClient, r: StompResponse ) =
  let id = r[ "ack" ]

  try:
    rowLength = parseInt(r["row-len"])
    activeNodesPerRow = to[seq[int]](r.payload)
  except CatchableError as e:
    echo "Couldn't parse! ", r.payload
    c.nack( id )
```

```
  var rows: seq[string]
  for i in 0 ..< activeNodesPerRow.len:
    rows.add($i)

  comboBoxRow.options = rows

  var columns: seq[string]
  if activeNodesPerRow.len > 0:
    for i in 0 ..< activeNodesPerRow[0]:
      columns.add($i)

  comboBoxColumn.options = columns
  c.ack(id)

proc messageLoop( c: StompClient, r: StompResponse ) =
  let id = r[ "ack" ]
  let messageType = parseInt(r["type"])

  if messageType == 3:
    echo "the cell value is: " & r.payload
    c.ack(id)
    return

  if (not observingNode):
    c.ack(id)
    return

  try:
    gameState = to[seq[int]](r.payload).distribute(rowLength, false)



    widthOffset = floorDiv(min(800,gameState[0].len), 2)
    heightOffset = floorDiv(min(800,gameState.len), 2)

    for i in 0 ..< min(800,gameState.len):
      for j in 0 ..< min(800,gameState[i].len):
        if (bool)gameState[i][j]:
```

```
          gameStateImage.canvas.setPixel(j, i, rgb(45, 220, 0))
        else:
          gameStateImage.canvas.setPixel(j, i, rgb(30, 30, 30))

    control1.forceRedraw()
  except CatchableError as e:
     let
       e = getCurrentException()
       msg = getCurrentExceptionMsg()
     echo "Got exception ", repr(e), " with message ", msg
     var h = readLineFromStdin("> ")
     c.nack( id )
     return

  c.ack(id)

proc initWorld() =

  echo "Choose the initialization plan: "
  echo "1. Random values"
  echo "2. Load values from file (values.txt)"

  var choice = "0"
  while choice != "1" and choice != "2":
    choice =  readLineFromStdin("> ")

  var values: seq[int]
  var matrixLength: int
  if choice == "2":
    let valueFile = open("./values.txt")
    defer: valueFile.close()
    values = to[seq[int]](valueFile.readLine())
    matrixLength = (int)sqrt((float)values.len)
    echo "matrix length= " & $matrixLength
  else:
     matrixLength = readLineFromStdin("Enter the size of the square matrix NxN. N=
").parseInt()
```

```
  var nodeNumber = readLineFromStdin("Enter gridworld size NxN (0 to calculate based on
memory) N= ").parseInt()
  var allowedMemoryUsage: float
  if nodeNumber == 0:
    allowedMemoryUsage = readLineFromStdin("Enter the percentage of free memory to be used:
 ").parseFloat()
    allowedMemoryUsage /= 100

  var layers:int = 0
  while layers < 2:
    layers =  readLineFromStdin("Enter the number of layers: ").parseInt()

  var headers: seq[ tuple[name:string, value:string] ]
  headers.add(("matrix-len", $matrixLength))
  headers.add(("node-num", $nodeNumber))
  headers.add(("layers", $layers))
  headers.add(("mem-use", $allowedMemoryUsage))

  client.send( "/topic/gatewaynodeinit", $$values, "text/plain", headers )



proc generateNodeData() {.thread.} =
  {.cast(gcsafe).}:

    let uri = "stomp://" & username & ":" & password & "@" & address & "/"

    client = newStompClient( socket, uri )

    # Attach callbacks
    client.connected_callback = connected
    client.connect

    initWorld()

    client.message_callback = infoMessage

    client.subscribe( "/topic/clientinfo", "client-individual" )
    client.wait_for_messages(false)
```

```
    client.message_callback = messageLoop

    client.unsubscribe( "/topic/clientinfo")



    client.wait_for_messages()




proc controlDrawn(event: DrawEvent) =
    gameCanvas = event.control.canvas
    # A shortcut

    gameCanvas.areaColor = rgb(30, 30, 30) # dark grey
    gameCanvas.fill()

    gameCanvas.textColor = rgb(0, 255, 0)
    gameCanvas.fontSize = 20
    gameCanvas.fontFamily = "Arial"



    var image_x = floorDiv(WINDOW_WIDTH, 2) - widthOffset
    var image_y = floorDiv(WINDOW_HEIGHT, 2) - heightOffset

    gameCanvas.drawImage(gameStateImage, image_x, image_y)

proc nodeDiscarded(event: CloseClickEvent) =
  observingNode = false
  client.unsubscribe( "/topic/clientmessage")
  window.dispose()

proc drawGraphics()  =
  #{.cast(gcsafe).}:

  window = newWindow()
  window.width = WINDOW_WIDTH
```

```
  window.height = WINDOW_HEIGHT

  control1 = newControl()



  control1.widthMode = WidthMode_Fill
  control1.heightMode = HeightMode_Fill

  window.add(control1)

  window.onCloseClick = nodeDiscarded

  control1.onDraw = controlDrawn

  gameStateImage = newImage()
  gameStateImage.resize(800, 800)
  for i in 0 ..< 800:
    for j in 0 ..< 800:
      gameStateImage.canvas.setPixel(i, j, rgb(30, 30, 30))

  window.show()

proc nodeSelected(event: ClickEvent) =
  if comboBoxColumn.options.len == 0 or comboBoxRow.options.len == 0: return

  observingNode = true

  drawGraphics()

  var row = comboBoxRow.index
  var column = comboBoxColumn.index



  var headers: seq[ tuple[name:string, value:string] ]
  headers.add( ("type", $2) )
  var payload: seq[int]
  payload.add(row)
```

```
  payload.add(column)

  client.send("/topic/clientrequest", $$payload, "text/plain", headers )

  var prefetch: seq[ tuple[name:string, value:string] ]
  prefetch.add( ("prefetch-count", $1) )
  client.subscribe( "/topic/clientmessage", "client-individual", "", prefetch )



proc rowChanged(event: ComboBoxChangeEvent) =
  var index = comboBoxRow.index
  var columns: seq[string]
  for i in 0 ..< activeNodesPerRow[index]:
    columns.add($i)

  comboBoxColumn.options = columns
  comboBoxColumn.index = 0



proc drawMenu() {.thread.}  =
  {.cast(gcsafe).}:
    app.init()

    var menuwidow = newWindow()
    menuwidow.width = 160
    menuwidow.height = 160

    var container1 = newLayoutContainer(Layout_Horizontal)

    var label1 = newLabel("Node row:      ")
    container1.add(label1)
    comboBoxRow = newComboBox()
    container1.add(comboBoxRow)

    comboBoxRow.onChange = rowChanged

    var container2 = newLayoutContainer(Layout_Horizontal)
```

```
    var label2 = newLabel("Node column: ")
    container2.add(label2)
    comboBoxColumn = newComboBox()
    container2.add(comboBoxColumn)

    var container3 = newLayoutContainer(Layout_Horizontal)

    var button = newButton("Show")
    container3.add(button)

    button.onClick = nodeSelected

    var containerMain = newLayoutContainer(Layout_Vertical)
    containerMain.add(container1)
    containerMain.add(container2)
    containerMain.add(container3)

    menuwidow.add(containerMain)

    menuwidow.show()

    app.run()

proc initialize() =

  let creds = open("./client/credentials.txt")
  defer: creds.close()
  address = creds.readLine()
  username = creds.readLine()
  password = creds.readLine()



proc main() =
  initialize()
  spawn generateNodeData()
  spawn drawMenu()
```

```
  sync()


main()
```