Πανεπιστημιο Πατρων
Τμημα Ηλεκτρολογων Μηχανικων και Τεχνολογιας Υπολογιστων
Τομεας Ηλεκτρονικης και Υπολογιστων

Υλοποιηση της τεχνικης της προωθησης σελιδων μνημης σε συσταδες υπολογιστων

Διπλωματικη Εργασια

Θωμα Ρεπαντη του Σπυριδωνος

Φοιτητη του Τμηματος
Ηλεκτρολογων Μηχανικων και Τεχνολογιας Υπολογιστων

Επιβλεπων: Καθηγητης **Θεοδωρος Σ. Παπαθεοδωρου**
Συνεπιβλεπων: Αναπληρωτης Καθηγητης **Δημητριος Ν. Σερπανος**

Αριθμος Διπλωματικης Εργασιας:
Νοεμβριος 2002

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η διπλωματική εργασία με θέμα:

Υλοποιηση της τεχνικης της προωθησης σελιδων μνημης σε συσταδες υπολογιστων

του φοιτητή του Τμήματος
Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Θωμά Ρεπαντή του Σπυρίδωνος
(Α.Μ. 4218)

παρουσιάσθηκε δημόσια και εξετάσθηκε στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις .

Ο Επιβλέπων

Καθηγητής Θ. Σ. Παπαθεοδώρου

Ο Διευθυντής του Τομέα

Καθηγητής Ε. Χούσος

Αριθμος Διπλωματικης Εργασιας:

Τιτλος:
Υλοποιηση της τεχνικης της προωθησης σελιδων μνημης σε συσταδες υπολογιστων

Φοιτητής: Θωμάς Σ. Ρεπαντής
Επιβλέπων: Καθηγητής Θεόδωρος Σ. Παπαθεοδώρου
Συνεπιβλέπων: Αναπληρωτής Καθηγητής Δημήτριος Ν. Σερπάνος

Συντομη Περιληψη:
Οι συστάδες υπολογιστών είναι μια σχετικά νέα και ελκυστική, λόγω του χαμηλού της κόστους και της ευελιξίας της, πλατφόρμα υπολογισμού υψηλών επιδόσεων. Ιδιαίτερα ενδιαφέρουσα προβάλλει η χρήση σε συστάδες υπολογιστών της αρχιτεκτονικής κατανεμημένης κοινής μνήμης, που παρέχοντας την ψευδαίσθηση της κοινής μνήμης, διευκολύνει το έργο του προγραμματιστή εφαρμογών. Οι κατανεμημένες κοινές μνήμες λογισμικού αν και πλεονεκτούν σε κόστος υλοποίησης δεν έχουν μέχρι σήμερα ιδιαίτερα καλές επιδόσεις συγκριτικά με ανταγωνιστικές πλατφόρμες.

Προκειμένου μία κατανεμημένη κοινή μνήμη λογισμικού να μπορεί να προσαρμόζεται σε μεταβαλλόμενους υπολογιστικούς πόρους, αλλά και στις εγγενείς υπολογιστικές και επικοινωνιακές ανάγκες της εκάστοτε εφαρμογής είναι πολύ χρήσιμη η υποστήριξη δυναμικής μετανάστευσης δεδομένων ανάμεσα στους σταθμούς, η οποία βοηθά στη βελτίωση της τοπικότητας στις προσβάσεις στη μνήμη και άρα της απόδοσης του συστήματος.

Η παρούσα διπλωματική εργασία ασχολείται με την ενσωμάτωση ενός μηχανισμού προώθησης σελίδων στην κατανεμημένη κοινή μνήμη λογισμικού JIAJIA . Αντικαταστάθηκε ο συντηρητικός μηχανισμός μετανάστευσης σελίδων του JIAJIA από έναν νέο, κατά τον οποίο ο πιο ισχυρός τροποποιητής μιας σελίδας μνήμης έχει την ευκαιρία να γίνει η νέα έδρα της. Κατά την υλοποίηση καταβλήθηκε ιδιαίτερη προσπάθεια για να διατηρηθεί απλό το πρωτόκολλο και για να ελαχιστοποιηθούν οι απαιτήσεις σε μετάδοση πληροφορίας μεταξύ των σταθμών. Ιδιαίτερη προσπάθεια απαιτήθηκε επίσης για την αντιμετώπιση αναγκών συγχρονισμού που εμφανίζονταν μόνο με το νέο πρωτόκολλο μετανάστευσης σελίδων.

Η σωστή λειτουργία του νέου μηχανισμού μετανάστευσης σελίδων επιβεβαιώθηκε

με τη λεπτομερή παρακολούθησή του σε κάθε στάδιο. Επιπλέον υλοποιήθηκαν μικρομετρο-
προγράμματα που προκαλούν τη διαδικασία της μετανάστευσης σελίδων, για να ελεγχ-
θεί η σωστή συμπεριφορά του μηχανισμού. Οι μετρήσεις που έγιναν με πραγματικές
εφαρμογές ήταν ενθαρρυντικές. Οι διαθέσιμες εφαρμογές δεν προσφέρονταν για την
ανάδειξη των πλεονεκτημάτων του μηχανισμού μετανάστευσης σελίδων, λόγω κυρί-
ως των περιορισμένων προσβάσεων σε απομακρυσμένες μνήμες, του μικρού ποσού
μεταφερόμενης πληροφορίας και των μικρών χρόνων εκτέλεσης. Ωστόσο η σύγκρ-
ιση του νέου μηχανισμού με τον παλαιότερο και με τη λειτουργία δίχως μηχανισμό
μετανάστευσης σελίδων επιβεβαιώνει ότι με μικρό επεξεργαστικό κόστος για το πι-
ο πολύπλοκο πρωτόκολλο καταφέρνουμε να μειώσουμε αρκετά το ποσό της μετα-
διδόμενης πληροφορίας. Η μείωση της επικοινωνιακής επιβάρυνσης, μαζί με την
ικανότητα για προσαρμογή στις μεταβαλλόμενες ανάγκες της εκάστοτε εφαρμογής
είναι και ο βασικός στόχος του νέου μηχανισμού μετανάστευσης σελίδων και το κύριο
πλεονέκτημα από τη χρήση αυτού.

Περίληψη

Τα τελευταία χρόνια για την επίλυση απαιτητικών προβλημάτων προτείνονται υλοποιήσεις που βασίζονται στην ιδέα της συνεργασίας πολλών εμπορικών υπολογιστών χαμηλού κόστους. Πρόκειται για τις συστάδες υπολογιστών, που έχουν πάρει τη θέση τους ανάμεσα στις κατηγορίες συστημάτων υψηλής επίδοσης, προσφέροντας ελκυστικό λόγο κόστους προς απόδοση, ευελιξία και παραμετροποιησιμότητα.

Ιδιαίτερο ενδιαφέρον παρουσιάζει η χρήση σε συστάδες υπολογιστών της αρχιτεκτονικής κατανεμημένης κοινής μνήμης. Πρόκειται για την αρχιτεκτονική συστημάτων που χρησιμοποιούν κατανεμημένη μνήμη, αλλά λειτουργούν σαν αυτή να ήταν κοινή και προσφέρουν αυτή την εικόνα στον προγραμματιστή, διευκολύνοντας έτσι το έργο του. Όταν μάλιστα η κατανεμημένη κοινή μνήμη υλοποιείται με λογισμικό, υπάρχει και ένα σαφές πλεονέκτημα κόστους υλοποίησης. Δυστυχώς το υψηλό κόστος επικοινωνίας σε συνδυασμό με το πολύπλοκο και απαιτητικό σε πόρους πρωτόκολλο διατήρησης της συνοχής της μνήμης καθιστούν τις επιδόσεις των συστημάτων αυτών μέτριες, σε σύγκριση με κατανεμημένες αρχιτεκτονικές που δεν προσφέρουν την ψευδαίσθηση της κοινής μνήμης στον προγραμματιστή.

Προκειμένου μία κατανεμημένη κοινή μνήμη λογισμικού να μπορεί να προσαρμόζεται σε μεταβαλλόμενους υπολογιστικούς πόρους, αλλά και στις εγγενείς

υπολογιστικές και επικοινωνιακές ανάγκες της εκάστοτε εφαρμογής είναι πολύ χρήσιμη η υποστήριξη δυναμικής μετανάστευσης δεδομένων ανάμεσα στους σταθμούς, η οποία βοηθά στη βελτίωση της τοπικότητας στις προσβάσεις στη μνήμη και άρα της απόδοσης του συστήματος.

Η παρούσα διπλωματική εργασία ασχολείται με τροποποιήσεις στην κατανεμημένη κοινή μνήμη λογισμικού JIAJIA . Μονάδα συνοχής στο JIAJIA είναι η σελίδα μνήμης. Κάθε σελίδα μνήμης έχει μια έδρα, αλλά μπορεί να διατηρείται και στη λανθάνουσα μνήμη άλλων σταθμών. Για την ασφαλή λειτουργία του JIAJIA , έγιναν οι κατάλληλες μετατροπές στον πηγαίο κώδικα ώστε οι αρχικές πληροφορίες που ανταλλάσσονται μεταξύ των σταθμών να μεταδίδονται κρυπτογραφημένες, χρησιμοποιώντας το πρωτόκολλο SSH.

Το JIAJIA παρέχει έναν στοιχειώδη μηχανισμό μετανάστευσης σελίδων μνήμης, τον οποίο και αντικαταστήσαμε με έναν πιο γενικό. Σύμφωνα με το συντηρητικό μηχανισμό μετανάστευσης σελίδων μνήμης του JIAJIA , οι σελίδες που εγγράφονται μόνο από έναν επεξεργαστή μεταξύ δύο φραγμάτων μεταναστεύουν στο μοναδικό αυτό εγγραφέα. Προσπαθώντας να διατηρήσουμε το πλεονέκτημα της μικρής επικοινωνιακής επιβάρυνσης, αλλά με στόχο έναν αλγόριθμο που να βρίσκει γενική εφαρμογή, υλοποιήσαμε ένα νέο μηχανισμό μετανάστευσης σελίδων μνήμης. Ο μηχανισμός αυτός, σε συμμόρφωση με το πρωτόκολλο συνοχής του JIAJIA , βασίζεται όπως και ο υπάρχων σε φράγματα. Σύμφωνα με αυτόν μεταναστεύουν και σελίδες που εγγράφονται και από περισσότερους από έναν σταθμούς μεταξύ δύο φραγμάτων. Η μετανάστευση γίνεται προς τον ισχυρότερο τροποποιητή της κάθε σελίδας, δηλαδή προς το σταθμό εκείνο που εφαρμόζει περισσότερες διαφορές σε αυτή. Προκειμένου να αποφευχθούν περιττές μεταναστεύσεις, η μετανάστευση λαμβάνει χώρα εφόσον οι απομακρυσμένες διαφορές ξεπερνούν κάποιο κατώφλι.

Το νέο πρωτόκολλο μετανάστευσης που υλοποιήθηκε απαιτείται να αντιμετωπίζει και καταστάσεις που με το προηγούμενο πρωτόκολλο δεν εμφαν΄-

ιζονταν, λόγω της πολύ περιορισμένης εφαρμογής του. Παράδειγμα αποτελεί η μεταφορά κάποιας σελίδας από την παλαιά στη νέα έδρα. Η διαδικασία αυτή απαιτεί επιπλέον συγχρονισμούς. Συγκεκριμένα, με την εισαγωγή επιπλέον μεταβλητών συγχρονισμού και με την υλοποίηση ενός προεξυπηρέτη των εισερχόμενων μηνυμάτων, ή με το διαχωρισμό της διαδικασίας ανανέωσης των πινάκων σελίδων και της μετακίνησης των σελίδων μνήμης σε δύο συγχρονιζόμενες επιμέρους διαδικασίες, εξασφαλίζεται ότι η νέα έδρα δε θα αναφερθεί στη νεοαποκτηθείσα σελίδα πριν αυτή έλθει πραγματικά από την παλαιά της έδρα και ότι η παλαιά έδρα δε θα αποαντιστοιχίσει κάποια σελίδα πριν τη στείλει στη νέα της έδρα.

Η σωστή λειτουργία του νέου μηχανισμού μετανάστευσης σελίδων επιβεβαιώθηκε με τη λεπτομερή παρακολούθησή του σε κάθε στάδιο. Επιπλέον υλοποιήθηκαν μικρομετροπρογράμματα που προκαλούν τη διαδικασία της μετανάστευσης σελίδων, για να ελεγχθεί η σωστή συμπεριφορά του μηχανισμού. Οι μετρήσεις που έγιναν με πραγματικές εφαρμογές ήταν ενθαρρυντικές. Οι διαθέσιμες εφαρμογές δεν προσφέρονταν για την ανάδειξη των πλεονεκτημάτων του μηχανισμού μετανάστευσης σελίδων. Ωστόσο η σύγκριση του νέου μηχανισμού με τον παλαιότερο και με τη λειτουργία δίχως μηχανισμό μετανάστευσης σελίδων επιβεβαιώνει ότι με μικρό επεξεργαστικό κόστος για το πιο πολύπλοκο πρωτόκολλο καταφέρνουμε να μειώσουμε αρκετά το ποσό της μεταδιδόμενης πληροφορίας. Η μείωση της επικοινωνιακής επιβάρυνσης, μαζί με την ικανότητα για προσαρμογή στις μεταβαλλόμενες ανάγκες της εκάστοτε εφαρμογής είναι και ο βασικός στόχος του νέου μηχανισμού μετανάστευσης σελίδων και το κύριο πλεονέκτημα από τη χρήση αυτού.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή Θεόδωρο Παπαθεοδώρου, για την ευκαιρία που μου έδωσε να εργασθώ σε ένα περιβάλλον υψηλού επιστημονικού επιπέδου όπως το Εργαστήριο Πληροφοριακών Συστημάτων Υψηλών Επιδόσεων, καθώς και για την εμπιστοσύνη που μου έδειξε.

Ευχαριστώ επίσης το συνεπιβλέποντα αναπληρωτή καθηγητή Δημήτριο Σερπάνο, για την υποστήριξή του και την πάντοτε συναρπαστική διδασκαλία του.

Θερμά ευχαριστώ το μεταπτυχιακό φοιτητή Χρήστο Αντωνόπουλο για την υποστήριξη και συμπαράσταση που μου παρείχε καθ' όλη τη διάρκεια της εκπόνησης της διπλωματικής αυτής εργασίας, τις συμβουλές, το χρόνο, την υπομονή και τον επαγγελματισμό του.

Ευχαριστώ επίσης το Δημήτριο Νικολόπουλο για την έρευνά του, που στάθηκε υπόδειγμα για τη διπλωματική αυτή εργασία.

Ακόμη ευχαριστώ τον καθηγητή Ευθύμιο Χούσο, για τη διδασκαλία, αλλά και τη στάση του ως καθηγητή, που έχει αποτελέσει υπόδειγμα για μένα έως σήμερα.

Τέλος ευχαριστώ το Λευτέρη Πολυχρονόπουλο, αλλά και όλα τα υπόλοιπα μέλη της ομάδας Παραλλήλων Συστημάτων του Εργαστηρίου Πληροφοριακών Συστημάτων Υψηλών Επιδόσεων για τις συμβουλές που μου παρείχαν.

Πάτρα, Νοέμβριος 2002
Θωμάς Ρεπαντής

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Οι συστάδες υπολογιστών και τα συστήματα κατανεμημένης κοινής μνήμης λογισμικού

Τα τελευταία χρόνια για την επίλυση απαιτητικών προβλημάτων προτείνονται υλοποιήσεις που βασίζονται στην ιδέα της συνεργασίας πολλών υπολογιστών. Πρόκειται για τις συστάδες υπολογιστών, που έχουν πάρει τη θέση τους ανάμεσα στις κατηγορίες συστημάτων υψηλής επίδοσης. Η ιδέα που κρύβεται πίσω από τις συστάδες υπολογιστών είναι η συνδυασμένη χρήση πολλών υπολογιστικών συστημάτων χαμηλού κόστους, λαμβάνοντας υπ΄ όψιν έννοιες που μας έχουν κληροδοτήσει οι περιοχές των παράλληλων και των κατανεμημένων συστημάτων. Σε σύγκριση με τις παραδοσιακές παράλληλες αρχιτεκτονικές, οι συστάδες υπολογιστών υστερούν συχνά σε επιδόσεις. Χρησιμοποιώντας όμως τεχνολογίες υλικού και λογισμικού ευρείας χρήσης κρατούν χαμηλό το κόστος υλοποίησης, με αποτέλεσμα να προσφέρουν έναν ελκυστικό λόγο κόστους προς απόδοση. Επιπλέον έχουν το πλεονέκτημα της ευελιξίας και παραμετροποιησιμότητας.

Συμβατικά οι παράλληλες αρχιτεκτονικές διαχωρίζονται ανάλογα με τον τρόπο υλοποίησης της μνήμης και συνεπώς και του τρόπου επικοινωνίας των επεξεργαστών σε αρχιτεκτονικές κατανεμημένης και αρχιτεκτονικές κοινής μνήμης. Ωστόσο έχουν γίνει αρκετά δημοφιλή και συστήματα που χρησιμοποιούν κατανεμημένη μνήμη, αλλά λειτουργούν σαν αυτή να ήταν κοινή και προσφέρουν αυτή την εικόνα στον προγραμματιστή. Τα συστήματα αυτά ονομάζονται συστήματα κατανεμημένης κοινής μνήμης και παρουσιάζει ενδιαφέρον η χρήση τους σε συστάδες υπολογιστών. Η εικόνα της κοινής μνήμης προσφέρει μεγάλη ευκολία στον προγραμματισμό τους. Όταν μάλιστα η κατανεμημένη κοινή μνήμη υλοποιείται με λογισμικό, υπάρχει και ένα σαφές πλεονέκτημα κόστους υλοποίησης. Δυστυχώς το υψηλό κόστος επικοινωνίας σε συνδυασμό με το πολύπλοκο και απαιτητικό σε πόρους πρωτόκολλο διατήρησης της συνοχής της μνήμης καθιστούν τις επιδόσεις των συστημάτων αυτών μέτριες, σε σύγκριση με κατανεμημένες αρχιτεκτονικές που δεν προσφέρουν την ψευδαίσθηση της κοινής μνήμης στον προγραμματιστή.

Η βασική ιδέα πίσω από την υλοποίηση του ενιαίου χώρου διευθύνσεων σε μία κατανεμημένη κοινή μνήμη λογισμικού είναι η μεσολάβηση ενός επιπέδου λογισμικού ανάμεσα στον προγραμματιστή και το υλικό. Το επίπεδο αυτό αναλαμβάνει να αντιστοιχίζει τις τοπικές μνήμες με τον κοινό εικονικό χώρο διευθύνσεων και αναλαμβάνει να διατηρεί τον τελευταίο συνεκτικό. Βασικό ρόλο σε ένα τέτοιο σύστημα έχουν επομένως το μοντέλο της συνέπειας της μνήμης και το πρωτόκολλο συνοχής της λανθάνουσας μνήμης. Αυτά είναι μεταξύ άλλων και χαρακτηριστικά κατηγοριοποίησης κατανεμημένων κοινών μνημών υλοποιημένων με λογισμικό.

1.2 Η σημασία της προώθησης σελίδων μνήμης στις συστάδες υπολογιστών

Προκειμένου μία κατανεμημένη κοινή μνήμη λογισμικού να μπορεί να προσαρμόζεται σε μεταβαλλόμενους υπολογιστικούς πόρους, αλλά και στις εγγενείς υπολογιστικές και επικοινωνιακές ανάγκες της εκάστοτε εφαρμογής είναι πολύ χρήσιμη η υποστήριξη δυναμικής μετανάστευσης εργασιών ανάμεσα στους σταθμούς. Η μετανάστευση εργασιών όμως είναι περισσότερο αποδοτική όταν συνδυάζεται και με δυναμική μετανάστευση των δεδομένων που αφορούν τις εργασίες αυτές. Η μετανάστευση δεδομένων μπορεί δε να βελτιώσει την απόδοση ακόμη και όταν δεν υποστηρίζεται μετανάστευση εργασιών, αυξάνοντας την τοπικότητα του συστήματος. Προωθώντας σελίδες μνήμης στους σταθμούς μιας συστάδας υπολογιστών που τις χρησιμοποιούν περισσότερο κάθε στιγμή, μειώνονται οι αργές λόγω διασυνδετικού δικτύου απομακρυσμένες προσβάσεις μνήμης και το σύστημα προσαρμόζεται δυναμικά στις ανάγκες και τις αλλαγές της εκάστοτε εφαρμογής.

1.3 Το JIAJIA και ο υπάρχων μηχανισμός μετανάστευσης σελίδων

Η παρούσα διπλωματική εργασία ασχολείται με τροποποιήσεις στην κατανεμημένη κοινή μνήμη λογισμικού JIAJIA . Πρόκειται για μια απλή και αποδοτική υλοποίηση που ακολουθεί το προγραμματιστικό μοντέλο Μοναδικής Ροής Προγράμματος και Πολλαπλής Ροής Δεδομένων. Μονάδα συνοχής στο JIAJIA είναι η σελίδα μνήμης. Κάθε σελίδα μνήμης έχει μια έδρα, αλλά μπορεί να διατηρείται και στη λανθάνουσα μνήμη άλλων σταθμών. Για την ασφαλή λειτουργία του JIAJIA , έγιναν οι κατάλληλες μετατροπές στον πηγαίο κώδικα

ώστε οι αρχικές πληροφορίες που ανταλλάσσονται μεταξύ των σταθμών να μεταδίδονται κρυπτογραφημένες, χρησιμοποιώντας το πρωτόκολλο SSH.

Το JIAJIA παρέχει έναν στοιχειώδη μηχανισμό μετανάστευσης σελίδων μνήμης, τον οποίο και αντικαταστήσαμε με έναν πιο γενικό. Σύμφωνα με το συντηρητικό μηχανισμό μετανάστευσης σελίδων μνήμης του JIAJIA , οι σελίδες που εγγράφονται μόνο από έναν επεξεργαστή μεταξύ δύο φραγμάτων μεταναστεύουν στο μοναδικό αυτό εγγραφέα. Αν και ο αλγόριθμος μετανάστευσης αυτός βρίσκει εφαρμογή σε λίγες περιπτώσεις, έχει ένα σημαντικό πλεονέκτημα: Δεν απαιτεί επιπλέον μηνύματα για τη μετανάστευση. Οι πληροφορίες που είναι απαραίτητες για την ενημέρωση των σταθμών για τη μετανάστευση φορτώνονται στο ήδη υπάρχον μήνυμα παραχώρησης φράγματος, ενώ η μετάδοση της μεταναστεύουσας σελίδας δεν απαιτείται, αφού ο μοναδικός της τροποποιητής, που γίνεται η νέα της έδρα, έχει ήδη ένα έγκυρο αντίγραφο.

1.4 Ο νέος μηχανισμός μετανάστευσης σελίδων που υλοποιήθηκε

Προσπαθώντας να διατηρήσουμε το πλεονέκτημα της μικρής επικοινωνιακής επιβάρυνσης, αλλά με στόχο έναν αλγόριθμο που να βρίσκει γενική εφαρμογή, υλοποιήσαμε ένα νέο μηχανισμό μετανάστευσης σελίδων μνήμης. Ο μηχανισμός αυτός, σε συμμόρφωση με το πρωτόκολλο συνοχής του JIAJIA , βασίζεται όπως και ο υπάρχων σε φράγματα. Σύμφωνα με αυτόν μεταναστεύουν και σελίδες που εγγράφονται και από περισσότερους από έναν σταθμούς μεταξύ δύο φραγμάτων. Η μετανάστευση γίνεται προς τον ισχυρότερο τροποποιητή της κάθε σελίδας, δηλαδή προς το σταθμό εκείνο που εφαρμόζει περισσότερες διαφορές σε αυτή. Προκειμένου να αποφευχθούν περιττές μεταναστεύσεις, η

μετανάστευση λαμβάνει χώρα εφόσον οι απομακρυσμένες διαφορές ξεπερνούν κάποιο κατώφλι.

Το νέο πρωτόκολλο μετανάστευσης που υλοποιήθηκε απαιτείται να αντιμετωπίζει και καταστάσεις που με το προηγούμενο πρωτόκολλο δεν εμφανίζονταν, λόγω της πολύ περιορισμένης εφαρμογής του. Συνοπτικά το πρωτόκολλο που υλοποιήθηκε έχει ως εξής: Κάθε σταθμός μετράει τα bytes των απομακρυσμένων διαφορών που εφαρμόζονται στις σελίδες που εδράζονται σε αυτόν. Φθάνοντας σε ένα φράγμα, ενημερώνει το διαχειριστή φράγματος για το ποιες σελίδες του τροποποιήθηκαν ισχυρά, καθώς και από ποιον σταθμό. Ο σταθμός αυτός πρόκειται να είναι η νέα έδρα της εκάστοτε σελίδας. Οι πληροφορίες αυτές φορτώνονται στο ήδη υπάρχον μήνυμα της αίτησης φράγματος. Ο διαχειριστής φράγματος λαμβάνει τις πληροφορίες μετανάστευσης όλων των σταθμών, τις συλλέγει και τις στέλνει σε όλους τους σταθμούς φορτωμένες στο επίσης ήδη υπάρχον μήνυμα της παραχώρησης φράγματος. Οι πληροφορίες μετανάστευσης εξετάζονται από κάθε σταθμό. Όλοι οι σταθμοί ενημερώνουν τον καθολικό πίνακα σελίδων τους. Επιπλέον η παλαιά έδρα κάθε σελίδας την αποδεσμεύει από τις εντός έδρας σελίδες της, ενώ η νέα έδρα την καταχωρεί στις εντός έδρας σελίδες της. Εάν η νέα έδρα ήταν ο μοναδικός εγγραφέας της σελίδας κατά το τελευταίο διάστημα μεταξύ φραγμάτων έχει ήδη τη σελίδα, οπότε και δε τη ζητά από την προηγούμενη έδρα. Σε διαφορετική περίπτωση τη ζητά από την προηγούμενη έδρα. Η διαδικασία αυτή απαιτεί επιπλέον συγχρονισμούς, που αντιμετωπίσθησαν με δύο εναλλακτικούς τρόπους.

Όπως είναι φανερό, έγινε κάθε προσπάθεια ώστε τα επιπλέον μηνύματα για τη μετανάστευση να είναι όσο το δυνατόν λιγότερα. Έτσι οι πληροφορίες μετανάστευσης φορτώνονται σε ήδη υπάρχοντα μηνύματα, ενώ η μετάδοση της μεταναστεύουσας σελίδας γίνεται μόνο όταν είναι αναγκαία.

1.5 Πειράματα, μετρήσεις και συμπεράσματα

Η σωστή λειτουργία του νέου μηχανισμού μετανάστευσης σελίδων επιβεβαιώθηκε με τη λεπτομερή παρακολούθησή του σε κάθε στάδιο και με υλοποιηθέντα μικρομετροπρογράμματα πρόκλησης της διαδικασίας της μετανάστευσης σελίδων. Οι μετρήσεις που έγιναν με πραγματικές εφαρμογές ήταν ενθαρρυντικές. Παρόλο που οι διαθέσιμες εφαρμογές δεν προσφέρονταν για την ανάδειξη των πλεονεκτημάτων του μηχανισμού μετανάστευσης σελίδων, η σύγκριση του νέου μηχανισμού με τον παλαιότερο και με τη λειτουργία δίχως μηχανισμό μετανάστευσης σελίδων επιβεβαιώνει ότι με μικρό επεξεργαστικό κόστος για το πιο πολύπλοκο πρωτόκολλο καταφέρνουμε να μειώσουμε αρκετά το ποσό της μεταδιδόμενης πληροφορίας. Η μείωση της επικοινωνιακής επιβάρυνσης, μαζί με την ικανότητα για προσαρμογή στις μεταβαλλόμενες ανάγκες της εκάστοτε εφαρμογής είναι και ο βασικός στόχος του νέου μηχανισμού μετανάστευσης σελίδων και το κύριο πλεονέκτημα από τη χρήση αυτού.

1.6 Η δομή του υπόλοιπου της διπλωματικής εργασίας

Η συνέχεια της παρούσας διπλωματικής εργασίας έχει ως εξής: Στο δεύτερο κεφάλαιο παρουσιάζεται μια επισκόπηση της συγκεκριμένης επιστημονικής περιοχής. Στο τρίτο κεφάλαιο παρουσιάζεται ο υπάρχων μηχανισμός μετανάστευσης σελίδων. Στη συνέχεια παρουσιάζεται αναλυτικά ο νέος μηχανισμός μετανάστευσης σελίδων και η υλοποίησή του. Στο τέταρτο κεφάλαιο παρουσιάζονται πειράματα με υλοποιηθέντα μικρομετροπρογράμματα και μετρήσεις με πραγματικές εφαρμογές και στο πέμπτο κεφάλαιο παρατίθενται συμπεράσματα.

Κεφάλαιο 2

Επισκόπηση της επιστημονικής περιοχής

2.1 Συστάδες υπολογιστών

2.1.1 Εφαρμογές των συστάδων υπολογιστών

Παρά τη ραγδαία εξέλιξη των υπολογιστών, εξακολουθούν να υπάρχουν υπολογιστικά προβλήματα που λόγω αυξημένων απαιτήσεων σε επεξεργαστική ισχύ, μνήμη και ταχύτητα εισόδου/εξόδου παραμένουν άλυτα ή δύσκολα επιλύσιμα. Το ερευνητικό πρόγραμμα High Performance Computing and Communications (HPCC — Υπολογισμός και Επικοινωνίες Υψηλής Επίδοσης) προσδιόρισε ορισμένες περιοχές εφαρμογών τέτοιου τύπου με σπουδαία οικονομική και κοινωνική σημασία [33]:

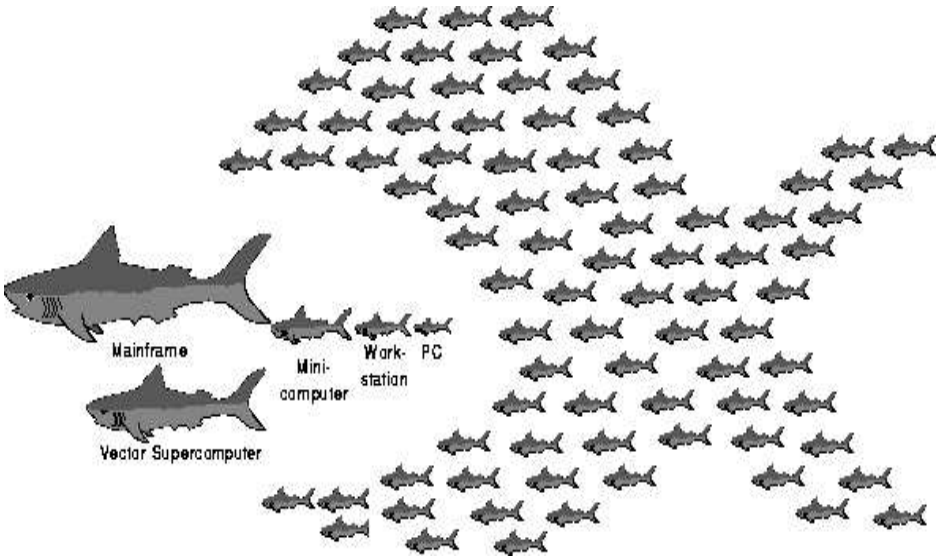
- Τεχνολογία μαγνητικής καταγραφής.
- Λογική σχεδίαση φαρμάκων.

- Μεταφορές υψηλής ταχύτητας.
- Χημική κατάλυση.
- Μοντελοποίηση ωκεανών.
- Μείωση όζοντος.
- Ψηφιακή ανατομία.
- Ατμοσφαιρική ρύπανση.
- Σχεδίαση πρωτεϊνικών δομών.
- Κατανόηση εικόνας.

Έχει καταστεί σαφές ότι οι υπάρχοντες ακολουθιακοί υπολογιστές δεν επαρκούν για την επίλυση τέτοιων προβλημάτων. Παραδοσιακά τέτοιου είδους προβλήματα αντιμετωπίζονταν με τη χρήση πολυεπεξεργαστικών υπολογιστικών συστημάτων, παράλληλων ή κατανεμημένων.

Τα τελευταία χρόνια ωστόσο κερδίζουν έδαφος υλοποιήσεις που βασίζονται στην ιδέα της συνεργασίας πολλών υπολογιστών για την επίλυση ενός προβλήματος, όπως καθιστά σαφές και το σχήμα 2.1. Παρουσιάζεται έτσι αυξημένο ενδιαφέρον για τις συστάδες (clusters) υπολογιστών [34], ως εναλλακτικά συστήματα υψηλής επίδοσης, εξαιτίας συγκεκριμένων πλεονεκτημάτων που αναλύονται στη συνέχεια.

Επιπρόσθετα και πέρα από εφαρμογές όπως οι παραπάνω που προέρχονται κύρια από το χώρο του επιστημονικού υπολογισμού, αποκτά ολοένα και περισσότερη αποδοχή η χρήση συστάδων υπολογιστών στην περιοχή των ενσωματωμένων συστημάτων πραγματικού χρόνου και ιδιαίτερα σε εφαρμογές διαστημικής, επεξεργασίας σήματος και τηλεπικοινωνιών [15]. Τέλος δε θα μπορούσε



Σχήμα 2.1: Η ιδέα της συνεργασίας πολλών υπολογιστών για την επίλυση προβλημάτων [5].

να παραλειφθεί η ολοένα και αυξανόμενη χρήση συστάδων υπολογιστών για την ανάκαμψη από σφάλματα (fail-over), την επίτευξη υψηλής διαθεσιμότητας (high availability) [26], καθώς και την εξισορρόπηση φορτίου (load balancing), ιδιότητες ιδιαίτερα χρήσιμες σε πολυσύχναστους δικτυακούς τόπους.

2.1.2 Ορισμός των συστάδων υπολογιστών

Συνδυάζοντας ιδέες που μας κληροδότησαν τα παράλληλα συστήματα, όπως η ομαδοποίηση πόρων υλικού (συχνά γενικού σκοπού), με έννοιες από τα κατανεμημένα συστήματα, όπως το αφαιρετικό μοντέλο περάσματος μηνυμάτων και χρησιμοποιώντας τις υπολογιστικές δυνατότητες χαμηλού κόστους που προσφέρει η βιομηχανία των προσωπικών υπολογιστών και των δικτύων οδηγηθήκαμε στη γέννηση των συστάδων υπολογιστών. Οι εμπορικές συστάδες σταθμών εργασίας ή προσωπικών υπολογιστών και τα συστήματα της τάξης Beowulf [2] αποτελούν σήμερα μια ταχύτατα αναπτυσσόμενη κατηγορία υπολογιστικών συστημάτων υψηλής επίδοσης.

Μια *εμπορική συστάδα (commodity cluster)* είναι ένα τοπικό υπολογιστικό σύστημα αποτελούμενο από μια ομάδα ανεξάρτητων υπολογιστών και ένα δίκτυο που τους διασυνδέει [31]. Μια συστάδα είναι τοπική με την έννοια ότι όλα τα επιμέρους συστήματα που την αποτελούν επιτηρούνται εντός μίας διαχειριστικής περιοχής. Οι υπολογιστικοί αυτοί κόμβοι είναι *εμπορικοί και άμεσα διαθέσιμοι (commercial off the shelf — COTS)*, ικανοί να λειτουργούν και ανεξάρτητα και γενικά του είδους που χρησιμοποιείται αυτόνομα για συνήθεις λειτουργίες. Ο κάθε κόμβος μπορεί να αποτελείται είτε από έναν μικροεπεξεργαστή είτε από πολλούς σε μια συμμετρική πολυεπεξεργαστική σύνθεση (symmetric multiprocessor — SMP). Το διασυνδετικό δίκτυο χρησιμοποιεί εμπορικές και άμεσα διαθέσιμές τεχνολογίες δικτύων, τοπικών (local area network — LAN) ή περιοχής συστήματος (systems area network — SAN), οι οποίες απαρτίζουν πολλές ανεξάρτητες δικτυακές δομές ή μια ιεραρχία τέτοιων. Το δίκτυο της συστάδας είναι αφιερωμένο στην ενοποίηση των υπολογιστικών της κόμβων και είναι διαχωρισμένο από το εξωτερικό της περιβάλλον.

Μια συστάδα μπορεί να είναι ρυθμισμένη έτσι ώστε να εξυπηρετεί διάφορους σκοπούς, όπως υψηλή υπολογιστική επίδοση για την επίλυση ενός προβλήματος, υψηλή χωρητικότητα ή διαμεταγωγή για ένα φορτίο διεργασιών, υψηλή διαθεσιμότητα (μέσω πλεονασμού κόμβων) ή υψηλό εύρος ζώνης (μέσω μεγάλου αριθμού δίσκων και καναλιών εισόδου/εξόδου). Ένα *συστήμα της τάξης Beowulf (Beowulf-class system)* [2] είναι μια συστάδα με κόμβους οι οποίοι είναι προσωπικοί υπολογιστές ή μικροί συμμετρικά πολυεπεξεργαστικοί προσωπικοί υπολογιστές, ενοποιημένοι με εμπορικά και άμεσα διαθέσιμα δίκτυα, τοπικά ή περιοχής συστήματος και φιλοξενούντες ένα λειτουργικό σύστημα ανά κόμβο, συνήθως ανοικτού κώδικα και τύπου Unix [31]. Στην πλειοψηφία των συστημάτων τέτοιου τύπου υπάρχει ένας κόμβος υπεύθυνος για την επαφή με τον υπόλοιπο κόσμο (front-end node) και η πρόσβαση στους υπόλοιπους υπολογιστές γίνεται μέσω αυτού. Μία *πλειάδα (constellation)* διαφέρει από μια εμπορική συστάδα

στο γεγονός ότι ο αριθμός των επεξεργαστών στους συμμετρικούς πολυεπεξεργαστές των κόμβων υπερβαίνει των αριθμό των συμμετρικών πολυεπεξεργαστών που αποτελούν το σύστημα, καθώς και στο γεγονός ότι το διασυνδετικό δίκτυο των συμμετρικά πολυεπεξεργαστικών κόμβων είναι πιθανά ειδικής τεχνολογίας και σχεδιασμού [31]. Όταν οι διασυνδεόμενοι υπολογιστές είναι σταθμοί εργασίας (συχνά μάλιστα όχι αποκλειστικά αφιερωμένοι στο έργο της συστάδας) μιλάμε για *Δίκτυο Σταθμών Εργασίας (Network Of Workstations — NOW)* [5]. Τέλος, αναφερόμενοι στη χρήση υπολογιστικών πόρων διαθέσιμων μέσω του Διαδικτύου μιλάμε για *πλέγματα (grids)* υπολογιστών [9]. Φυσικά οι παραπάνω ορισμοί έχουν ως στόχο να αποσαφηνίσουν τις σχετικές έννοιες και όχι να αποκλείσουν ειδικές διατάξεις, που πιθανά επίσης εντάσσονται στη γενικότερη περιοχή των συστάδων υπολογιστών.

2.1.3 Πλεονεκτήματα και μειονεκτήματα των συστάδων υπολογιστών

Λαμβάνοντας κανείς υπ' όψιν του τις παραδοσιακές προσεγγίσεις από τους χώρους της παράλληλης επεξεργασίας και των παράλληλων αρχιτεκτονικών θα μπορούσε να καταλογίσει αρκετά μειονεκτήματα στις συστάδες υπολογιστών. Η εμπειρία έχει δείξει πόσο κρίσιμοι παράγοντες είναι η καθυστέρηση και το εύρος ζώνης του διασυνδετικού δικτύου, πόσο χρήσιμη είναι η κοινόχρηστη μνήμη και πόσο αναγκαίο το ελαφρύ λογισμικό ελέγχου. Σε όλα τα παραπάνω οι συστάδες υπολογιστών υστερούν σε σύγκριση με συμβατικά παράλληλα συστήματα [31]: Το διασυνδετικό δίκτυο είναι συνήθως χαμηλών —συγκριτικά με αυτά των παράλληλων συστημάτων— επιδόσεων, η κοινή μνήμη, ο καθολικός συγχρονισμός και η διατήρηση της συνοχής της κρυφής μνήμης υλοποιούνται με λογισμικό, υστερώντας σε επιδόσεις σε σχέση με αντίστοιχες υλοποιήσεις με υλικό, ενώ τέλος το πλήρες και αυτόνομο λειτουργικό σύστημα που ελέγχει κάθε κόμβο καταναλώνει περισσότερη μνήμη και αντιδρά πιο αργά από ειδικού

τύπου λογισμικό που μπορεί να ελέγχει κάθε κόμβο σε άλλου τύπου υλοποιήσεις. Τα παραπάνω μειονεκτήματα πράγματι καθιστούν τις συστάδες υπολογιστών ακατάλληλες για ορισμένες εφαρμογές. Η πράξη όμως αποδεικνύει πως σε πάρα πολλές περιπτώσεις και χρησιμοποιώντας συχνά νέες αλγοριθμικές τεχνικές που δεν εξαρτώνται τόσο από τη γρήγορη επικοινωνία των κόμβων οι συστάδες υπολογιστών μπορούν να ανταγωνιστούν τις επιδόσεις παράλληλων συστημάτων πολύ υψηλότερου κόστους.

Ένα σημαντικό πλεονέκτημα των συστάδων υπολογιστών είναι ότι χρησιμοποιούν τεχνολογίες υλικού και λογισμικού ευρείας χρήσης. Τόσο τα δίκτυα σταθμών εργασίας, όσο και οι συστάδες προσωπικών υπολογιστών της τάξης Beowulf άνθισαν διότι δεν απαιτούσαν δαπανηρά και μακροπρόθεσμα σχέδια ανάπτυξης πριν αρχίσουν να αποδίδουν. Το χαρακτηριστικό αυτό οδηγεί προφανώς και σε ένα σαφές πλεονέκτημα κόστους. Συνεπώς ο λόγος κόστους προς απόδοση των συστάδων υπολογιστών μπορεί να υπερτερεί εκείνου των παραδοσιακών υπερυπολογιστών για εφαρμογές που δεν απαιτούν επικοινωνία υψηλού εύρους ζώνης και χαμηλής καθυστέρησης [11].

Άλλο σημείο υπεροχής των εμπορικών συστάδων υπολογιστών έναντι των κλασσικών παράλληλων μηχανών είναι η ευελιξία και η παραμετροποιησιμότητα που οι πρώτες προσφέρουν. Ο αριθμός των κόμβων, η χωρητικότητα μνήμης ανά κόμβο, ο αριθμός των επεξεργαστών ανά κόμβο και η τοπολογία του διασυνδετικού δικτύου είναι δομικές παράμετροι που μπορούν να καθορισθούν με μεγάλη λεπτομέρεια και για κάθε συστάδα ξεχωριστά, χωρίς επιπρόσθετο κόστος λόγω εξειδικευμένων επιλογών. Επιπλέον η δομή του συστήματος μπορεί να αλλάζει ή αυτό να επεκτείνεται, αξιοποιώντας την ήδη υπάρχουσα υποδομή. Ο εκτεταμένος αυτός έλεγχος στη δομή του συστήματος ωφελεί τόσο τους χρήστες, που μπορούν να έχουν κάθε στιγμή αυτό που επιθυμούν, όσο και τους κατασκευαστές, που μπορούν να καλύπτουν ένα ευρύ φάσμα απαιτήσεων επιδόσεων ή κόστους.

Τέλος οι εμπορικές συστάδες υπολογιστών μπορούν να ενσωματώνουν γρήγορα τεχνολογικές εξελίξεις που χρησιμοποιούνται μαζικά. Νέοι τύποι επεξεργαστών, μνημών, δίσκων και δικτύων κατασκευάζονται διαρκώς για τους προσωπικούς υπολογιστές, επιτρέποντας και στις συστάδες να επωφελούνται από τις προόδους αυτές. Οι συστάδες επίσης επωφελούνται από πτώσεις τιμών λόγω της μαζικής παραγωγής των εξαρτημάτων προσωπικών υπολογιστών. Συνολικά θα λέγαμε πως ακόμα και αν οι συστάδες υπολογιστών δεν επιτυγχάνουν πάντα επιδόσεις αντάξιες κορυφαίων παράλληλων μηχανών το χαμηλό τους κόστος και η ευελιξία που προσφέρουν τις καθιστά σαφώς ανταγωνιστικές.

2.2 Συστήματα κατανεμημένης κοινής μνήμης

2.2.1 Ορισμός των συστημάτων κατανεμημένης κοινής μνήμης

Ένας συμβατικός διαχωρισμός των παράλληλων αρχιτεκτονικών γίνεται συχνά με βάση την υλοποίηση της μνήμης και συνεπώς του τρόπου επικοινωνίας των επεξεργαστών [32]. Έτσι μιλάμε για αρχιτεκτονικές κατανεμημένης (distributed) μνήμης και για αρχιτεκτονικές κοινής (shared) μνήμης. Στις περιπτώσεις κατανεμημένης μνήμης κάθε επεξεργαστής έχει τη δική του αποκλειστικά ιδιωτική μνήμη και οι μεταβλητές που βρίσκονται σε αυτή δε μοιράζονται με άλλους επεξεργαστές. Σε αυτές τις περιπτώσεις, η επικοινωνία μεταξύ επεξεργαστών γίνεται με πέρασμα μηνυμάτων. Στις περιπτώσεις κοινής μνήμης οι επεξεργαστές έχουν πρόσβαση σε κοινόχρηστη μνήμη, στην οποία καταχωρούν και από την οποία διαβάζουν μεταβλητές, με αποτέλεσμα και να επικοινωνούν κατ' αυτόν τον τρόπο. Στην τυπική περίπτωση, η πρόσβαση σε όλη την κοινή μνήμη είναι ισότιμη για όλους τους επεξεργαστές, με την έννοια ότι όλοι έχουν πρόσβαση σε όλα τα τμήματα της κοινής μνήμης και με ίδιους χρόνους

πρόσβασης, οπότε και μιλάμε για ομοιόμορφη πρόσβαση στη μνήμη (Uniform Memory Access — UMA). Συμβαίνει όμως συχνά να υπάρχουν και επιπλέον ιδιωτικές μνήμες ή άλλες ανισότητες σε προσβάσεις τμημάτων μνήμης. Έτσι όταν έχουμε τμήματα κοινής μνήμης στα οποία η πρόσβαση είναι ταχύτερη για συγκεκριμένους επεξεργαστές και σε σύγκριση με άλλα, απομακρυσμένα τμήματα μιλάμε για ανομοιόμορφη πρόσβαση στη μνήμη (Non Uniform Memory Access — NUMA). Η ανομοιομορφία στην υλοποίηση της πρόσβασης στη μνήμη δεν αντανακλάται ωστόσο στον προγραμματισμό των μηχανών αυτών, με την έννοια ότι όλα τα τμήματα της μνήμης παρουσιάζονται ισότιμα και ομοιόμορφα προσβάσιμα στον προγραμματιστή, χωρίς αυτό να εξασφαλίζεται απαραίτητα από την υποκείμενη υποδομή.

Πέρα όμως από το συμβατικό διαχωρισμό σε αρχιτεκτονικές κατανεμημένης και κοινής μνήμης, έχουν γίνει αρκετά δημοφιλή και συστήματα που χρησιμοποιούν κατανεμημένη μνήμη, αλλά λειτουργούν σαν αυτή να ήταν κοινή και προσφέρουν την εικόνα κοινής μνήμης στον προγραμματιστή. Τα συστήματα αυτά ονομάζονται συστήματα *κατανεμημένης κοινής μνήμης (distributed shared memory — DSM)* ή αλλιώς συστήματα *κοινής εικονικής μνήμης (shared virtual memory — SVM)* . Στην περιπτώση αυτών των συστημάτων κάθε επεξεργαστής έχει τη δική του φυσική μνήμη, η οποία όμως είναι κοινόχρηστη με την έννοια ότι όλοι οι επεξεργαστές μπορούν να τη γράφουν και να τη διαβάζουν και έτσι να επικοινωνούν μέσω κοινών μεταβλητών. Συνήθως μόνο ένα τμήμα της τοπικής μνήμης του κάθε επεξεργαστή είναι κοινόχρηστο και υπάρχει έτσι και τοπική ιδιωτική μνήμη για καθέναν.

2.2.2 Πλεονεκτήματα και μειονεκτήματα των συστημάτων κατανεμημένης κοινής μνήμης

Τα συστήματα κατανεμημένης κοινής μνήμης γνωρίζουν απήχηση, γιατί συνδυάζουν την επεκτασιμότητα/κλιμακωσιμότητα και το χαμηλό κόστος (λόγω

απλότητας υλοποίησης) των συστημάτων κατανεμημένης μνήμης με την προγραμματιστική ευκολία που παρέχει ο ενιαίος χώρος διευθύνσεων των συστημάτων κοινής μνήμης.

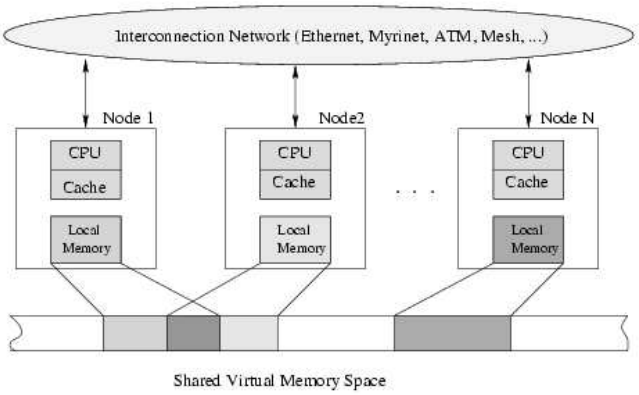
Φυσικά τα πλεονεκτήματα δεν έρχονται χωρίς κόστος και έτσι έχουμε συχνά προβλήματα απόδοσης, που σχετίζονται κύρια με το υψηλό επιπλέον κόστος για την κλήση και την επεξεργασία του πρωτοκόλλου διατήρησης της συνοχής, με το υψηλό επιπλέον κόστος της επικοινωνίας και με τη μεγάλη διαμοίραση στην επικοινωνία και τη συνοχή της μνήμης. Θα λέγαμε ωστόσο πως τα ήδη αναφερθέντα πολύ σημαντικά θετικά σημεία των συστημάτων κατανεμημένης κοινής μνήμης τα καθιστούν σοβαρές προτάσεις, ακόμα και αν οι επιδόσεις τους δεν είναι πάντα οι αναμενόμενες.

2.3 Συστήματα κατανεμημένης κοινής μνήμης υλοποιημένα με λογισμικό

2.3.1 Ορισμός των συστημάτων κατανεμημένης κοινής μνήμης υλοποιημένων με λογισμικό

Τα συστήματα κατανεμημένης κοινής μνήμης μπορούν να χωριστούν σε δύο βασικές κατηγορίες ανάλογα με τον τρόπο υλοποίησής τους: Συστήματα με ενιαίο χώρο διευθύνσεων υλοποιημένο με υλικό και συστήματα με ενιαίο χώρο διευθύνσεων υλοποιημένο με λογισμικό.

Ένα σύστημα κατανεμημένης κοινής μνήμης υλοποιημένο με λογισμικό (ή για συντομία σύστημα *κατανεμημένης κοινής μνήμης λογισμικού (Software DSM, SWDSM ή SDSM)*) αποτελείται από έναν ενιαίο χώρο διευθύνσεων,



Σχήμα 2.2: Η βασική ιδέα μιας κατανεμημένης κοινής μνήμης υλοποιημένης με λογισμικό [28].

τον οποίο μοιράζεται ένας αριθμός επεξεργαστών, όπως φαίνεται και στο σχήμα 2.2. Οποιοσδήποτε επεξεργαστής έχει απ΄ ευθείας πρόσβαση σε οποιαδήποτε θέση μνήμης. Στην περίπτωση των συστημάτων κατανεμημένης κοινής μνήμης υλοποιημένων με λογισμικό η ψευδαίσθηση της κοινής μνήμης παρέχεται στον προγραμματιστή από ένα επίπεδο λογισμικού, υλοποιημένο είτε ως μέρος του πυρήνα του λειτουργικού συστήματος, είτε ως μια βιβλιοθήκη χρόνου εκτέλεσης με την οποία συνδέονται οι εφαρμογές. Το επίπεδο λογισμικού αυτό πέραν της αντιστοίχισης μεταξύ των τοπικών μνημών και του κοινού εικονικού χώρου διευθύνσεων αναλαμβάνει να διατηρεί το χώρο διευθύνσεων συνεκτικό κάθε στιγμή. Έτσι η τιμή που επιστρέφει μια λειτουργία ανάγνωσης σε μια διεύθυνση είναι (στην απλούστερη περίπτωση) η τιμή που αποθήκευσε η πιο πρόσφατη λειτουργία εγγραφής στη συγκεκριμένη διεύθυνση. Προγράμματα εφαρμογών μπορούν να χρησιμοποιούν την κοινή εικονική μνήμη όπως μια παραδοσιακή εικονική μνήμη, μόνο που οι διεργασίες μπορούν να εκτελούνται σε διαφορετικούς επεξεργαστές παράλληλα.

2.3.2 Ανάλυση της λειτουργίας των συστημάτων κατανεμημένης κοινής μνήμης υλοποιημένων με λογισμικό

Σε ένα κλασσικό σύστημα κατανεμημένης κοινής μνήμης υλοποιημένο με λογισμικό, που ακολουθεί ένα ακολουθιακό μοντέλο συνέπειας της μνήμης, ο κοινός εικονικός χώρος διευθύνσεων χωρίζεται σε σελίδες. Οι σελίδες αυτές μπορούν να αντιγράφονται και να μετακινούνται μεταξύ επεξεργαστών όποτε αυτό απαιτείται, όπως συμβαίνει με τις γραμμές της λανθάνουσας (ή αλλιώς κρυφής — cache) μνήμης στα συστήματα κατανεμημένης κοινής μνήμης υλοποιημένα με υλικό. Κάθε σελίδα έχει συνήθως κατάσταση είτε μόνο - ανάγνωσης (read-only), είτε ανάγνωσης - εγγραφής (read-write) (οπότε και είναι μοναδικό το αντίγραφο της σελίδας στο σύστημα), είτε άκυρη (invalid). Το επίπεδο λογισμικού που υλοποιεί την κατανεμημένη κοινή μνήμη βλέπει την τοπική μνήμη σαν μια μεγάλη λανθάνουσα μνήμη της κοινής εικονικής μνήμης για τον αντίστοιχο επεξεργαστή και τη διαχειρίζεται με πλήρως συσχετιστικό τρόπο και ανάλογα με τον καταμερισμό σελίδων. Έτσι, όπως και η παραδοσιακή εικονική μνήμη, η κοινή μνήμη υπάρχει μόνο εικονικά. Μια αναφορά στη μνήμη προκαλεί ένα σφάλμα σελίδας όταν η σελίδα δε βρίσκεται εκείνη τη στιγμή στη φυσική μνήμη του επεξεργαστή που τη ζήτησε. Τότε το επίπεδο λογισμικού που υλοποιεί την κατανεμημένη κοινή μνήμη φέρνει τη σελίδα από τη μνήμη κάποιου άλλου επεξεργαστή. Στην περίπτωση που η σελίδα αυτή έχει αντίγραφα και σε άλλους επεξεργαστές τα αντίστοιχα επίπεδα λογισμικού συνεργάζονται για να διατηρήσουν τη μνήμη συνεκτική.

Βασική δυσκολία στην κατασκευή ενός συστήματος κατανεμημένης κοινής μνήμης λογισμικού είναι η επίλυση του προβλήματος της συνοχής της μνήμης, του προβλήματος δηλαδή του πως να γίνουν γνωστές εγκαίρως οι νέες τιμές στους επεξεργαστές που τις ζητάνε. Το μοντέλο της συνέπειας της μνήμης και το πρωτόκολλο της συνοχής της λανθάνουσας μνήμης είναι οι δυο σημαντικές

παράμετροι του προβλήματος αυτού. Το *μοντέλο της συνέπειας της μνήμης (memory consistency model)* καθορίζει πότε τα τροποποιημένα δεδομένα πρέπει να είναι εμφανή σε άλλους επεξεργαστές, ενώ το *πρωτόκολλο της συνοχής της λανθάνουσας μνήμης (cache coherence protocol)* καθορίζει ποιες τιμές πρέπει να είναι εμφανείς σε άλλους επεξεργαστές [28].

2.3.3 Κατηγοριοποίηση συστημάτων κατανεμημένης κοινής μνήμης λογισμικού

Μπορούμε να εξετάσουμε τα συστήματα κατανεμημένης κοινής μνήμης υλοποιημένα με λογισμικό από διάφορες απόψεις, όπως: Μεθόδος υλοποίησης, οργάνωση μνήμης, μοντέλο συνέπειας μνήμης, πρωτοκόλλο συνοχής λανθάνουσας μνήμης, λεπτότητα του καταμερισμού, συμμετρία του πρωτοκόλλου συνέπειας ή διεπαφή με τον προγραμματιστή εφαρμογών (Application Programming Interface — API). Μερικές αντιπροσωπευτικές κατανεμημένες κοινές μνήμες λογισμικού με τα κυριότερα χαρακτηριστικά τους φαίνονται στον πίνακα 2.1. Μια καλή κατηγοριοποίηση κατανεμημένων κοινών μνημών λογισμικού υπάρχει επίσης στο [16].

Μέθοδοι υλοποίησης

Όπως ίσως είναι αναμενόμενο, οι υλοποιήσεις κατανεμημένων κοινών μνημών που υλοποιούνται σε λογισμικό δεν καταφέρνουν να συναγωνισθούν εκείνες που υλοποιούνται σε υλικό σε επιδόσεις (σε συνθήκες υψηλού φόρτου) και ιδιαίτερα σε κλιμακωσιμότητα [6]. Λόγω όμως του χαμηλότερου κόστους υλοποίησης αποτελούν σοβαρή εναλλακτική πρόταση. Οι υλοποιήσεις σε λογισμικό βασίζονται σε τροποποιήσεις που μπορούν να λάβουν χώρα σε διάφορα επίπεδα, από τον πυρήνα του λειτουργικού συστήματος, μέχρι κάποιο μεταγλωττιστή ή

ακόμη και σε βιβλιοθήκες χρηστών (χρόνου εκτέλεσης). Δε λείπουν τέλος οι κατανεμημένες κοινές μνήμες λογισμικού που υποστηρίζονται από υλικό.

Οργάνωση μνήμης

Συνήθως η εικονική κοινή μνήμη μιας κατανεμημένης κοινής μνήμης λογισμικού είναι οργανωμένη με τρόπο που μοιάζει με COMA (Cache Only Memory Access): Η τοπική μνήμη κάθε σταθμού θεωρείται σαν μια μεγάλη λανθάνουσα μνήμη και σελίδες μπορούν να αντιγράφονται και να μετακινούνται ανάλογα με τις απαιτήσεις. Καθώς οι σελίδες αλλάζουν ιδιοκτήτη μειώνονται τα σφάλματα σελίδας, αλλά είναι απαραίτητος ένας μηχανισμός για την ανεύρεση των δεδομένων που δεν υπάρχουν τοπικά. Ένας άλλος τρόπος οργάνωσης της εικονικής κοινής μνήμης είναι εκείνος που μοιάζει με ccNUMA (cache coherent Non Uniform Memory Access): Σε αυτή την περίπτωση κάθε σελίδα έχει μια σταθερή έδρα, από την οποία ζητείται όταν χρειασθεί.

Μοντέλο συνέπειας μνήμης

Το μοντέλο συνέπειας μνήμης καθορίζει την εικόνα που έχει ο προγραμματιστής για το σύστημα και συνήθως επιβάλλει περιορισμούς στη σειρά πρόσβασης σε μεταβλητές τμημάτων της κοινής μνήμης. Ένα αυστηρό μοντέλο όπως η ακολουθιακή συνέπεια (sequential consistency — SC) είναι εύκολο στον προγραμματισμό, αλλά χαμηλό σε επιδόσεις, εξαιτίας και του προβλήματος της ψευδοδιαμοίρασης (false sharing) δεδομένων [17]. Έτσι προτιμούνται μοντέλα χαλαρής συνέπειας μνήμης, στα οποία οι ενημερώσεις τιμών αναβάλλονται μέχρι κάποια σημεία συγχρονισμού, ώστε να μειωθεί η επικοινωνιακή επιβάρυνση [25]. Ένα από τα σημαντικότερα τέτοια μοντέλα είναι η συνέπεια αδρανούς απελευθέρωσης (Lazy Release Consistency — LRC) [17].

Πρωτόκολλο συνοχής λανθάνουσας μνήμης

Η ύπαρξη πολλών αντιγράφων σελίδων σε λανθάνουσες μνήμες επιβάλλει ένα μηχανισμό ειδοποίησης όλων των κόμβων για τυχόν αλλαγές. Μία νέα τιμή μπορεί να διαδίδεται είτε ακυρώνοντας είτε ενημερώνοντας ένα αντίγραφο, οπότε τα αντίστοιχα πρωτόκολλα ειναι το write-invalidate και το write-update. Εκτός από τις ανυπόμονες eager-invalidate και eager-update, έχουν προταθεί και πολλές ιδέες χαλαρής εξασφάλισης συνοχής, προκειμένου η διάδοση των νέων τιμών να γίνεται πιο αποδοτικά. Ανάμεσά τους είναι και τα εξής πρωτόκολλα: lazy-invalidate, lazy-hybrid [17] [8]. Για τη διατήρηση της συνοχής δύο είναι οι βασικές μέθοδοι: πρωτόκολλα ανίχνευσης (snoopy protocols) και πρωτόκολλα βασισμένα σε κατάλογο (directory-based protocols). Στις κατανεμημένες κοινές μνήμες αποκλειστικά βασισμένες σε λογισμικό συνήθως χρησιμοποιείται το δεύτερο, πιθανά με κάποιες παραλλαγές, παρόλο που περιορίζει την κλιμακωσιμότητα του συστήματος λόγω της πολυπλοκότητάς του. Η χρήση πρωτοκόλλων ανίχνευσης δεν είναι διαδεδομένη, διότι απαιτεί υποστήριξη από το υλικό.

Λεπτότητα του καταμερισμού

Η λεπτότητα του καταμερισμού (granularity) αναφέρεται στο μέγεθος της μονάδας διαμοίρασης. Συνήθως αυτή είναι λέξη, σελίδα ή κάποια πολύπλοκη δομή δεδομένων (αντικείμενο) [3]. Ενώ οι κατανεμημένες κοινές μνήμες υλικού χρησιμοποιούν συνήθως τη γραμμή της λανθάνουσας μνήμης ως μονάδα συνοχής, οι κατανεμημένες κοινές μνήμες λογισμικού χρησιμοποιούν συνήθως τη σελίδα μνήμης. Το μέγεθος της μονάδας διαμοίρασης επηρεάζει την επικοινωνία, την ψευδοδιαμοίραση και το μέγεθος του καταλόγου.

Συμμετρία του πρωτοκόλλου συνέπειας

Το πρωτόκολλο συνέπειας της μνήμης χαρακτηρίζεται συμμετρικό όταν όλοι οι επεξεργαστές που προσπελαύνουν κοινούς πόρους αντιμετωπίζονται ισότιμα. Αντίθετα σε ένα ασύμμετρο πρωτόκολλο υπάρχει η έννοια της έδρας ή του διαχειριστή για κάθε κοινό πόρο. Σε ένα ασύμμετρο πρωτόκολλο η χρήση κάποιου πόρου από την έδρα του γίνεται με μικρότερο κόστος από ότι από άλλους επεξεργαστές. Καλές επιδόσεις των ασύμμετρων συστημάτων επιτυγχάνονται όταν η ασυμμετρία του πρωτοκόλλου είναι αντίστοιχη με την ασυμμετρία της εφαρμογής [19]. Αυτό μπορεί να εξασφαλίζεται και με τη δυναμική αλλαγή έδρας των πόρων, με βάση το πρότυπο πρόσβασης σε αυτούς. Οι μεταναστεύσεις αυτές μπορούν να αφορούν είτε εργασίες είτε δεδομένα [22].

Διεπαφή με τον προγραμματιστή εφαρμογών

Γενικά η διεπαφή με τον προγραμματιστή εφαρμογών στα παράλληλα συστήματα ακολουθεί συνήθως είτε το παράδειγμα της κοινής μνήμης, είτε το παράδειγμα του περάσματος μηνυμάτων. Στο πρώτο ο προγραμματιστής δεν ασχολείται με την κατανομή των δεδομένων, μετακινήσεις αυτών ή επικοινωνιακούς μηχανισμούς. Αντίθετα στο δεύτερο το σύστημα απαιτεί το ρητό ορισμό τέτοιων παραμέτρων, παρέχοντας στον προγραμματιστή συναρτήσεις επικοινωνίας για το σκοπό αυτό. Η διαδικασία αυτή αν και μπορεί να δώσει καλύτερες επιδόσεις είναι ιδιαίτερα επίπονη. Συνήθως οι κατανεμημένες κοινές μνήμες λογισμικού ακολουθούν το προγραμματιστικό μοντέλο των παραδοσιακών συστημάτων κοινής μνήμης, που μοιάζει αρκετά με το ακολουθιακό. Ωστόσο η ακριβής διεπαφή με τον προγραμματιστή εξαρτάται άμεσα από το μοντέλο συνέπειας της μνήμης και τη μέθοδο υλοποίησης. Για παράδειγμα μπορεί να εισάγονται νέα χαρακτηριστικά σε γνωστές γλώσσες προγραμματισμού, αν η υλοποίηση

έγινε τροποποιώντας κάποιο μεταγλωττιστή, ή να χρησιμοποιούνται κλήσεις νέων συναρτήσεων, αν η υλοποίηση έγινε ως βιβλιοθήκη χρόνου εκτέλεσης.

2.3.4 Πλεονεκτήματα και μειονεκτήματα των συστημάτων κατανεμημένης κοινής μνήμης υλοποιημένων με λογισμικό

Τόσο στην περίπτωση των συστημάτων κατανεμημένης κοινής μνήμης υλοποιημένων με υλικό, όσο και στην περίπτωση αυτών που υλοποιούνται με λογισμικό η κατανομή των επεξεργαστών και της μνήμης προκαλεί ανομοιόμορφη πρόσβαση στη μνήμη, μιας και απομακρυσμένες προσβάσεις στη μνήμη καθυστερούν πολύ περισσότερο από αντίστοιχες τοπικές. Η διαφύλαξη δεδομένων σε λανθάνουσες μνήμες περιπλέκει ακόμη περισσότερο την κατάσταση, αφού πολλά αντίγραφα της ίδιας περιοχής μνήμης εμποδίζουν την ατομικότητα των προσβάσεων στη μνήμη. Ως αποτέλεσμα της έλλειψης ομοιομορφίας και ατομικότητας στην πρόσβαση στη μνήμη έχουμε επιπλέον πολυπλοκότητα, προκειμένουν να διατηρηθεί η ορθότητα (correctness) του συστήματος.

Σκοπός ενός καλού συστήματος κατανεμημένης κοινής μνήμης είναι να επιτυγχάνει καλές επιδόσεις σε ένα μεγάλο εύρος παράλληλων προγραμμάτων, χωρίς να απαιτούνται αλλαγές σε αυτά (ή με μικρές τροποποιήσεις αντίστοιχων ακολουθιακών προγραμμάτων) και γενικά χωρίς να απαιτείται από τον προγραμματιστή εφαρμογών να λαμβάνει υπ' όψιν του τη διαμοίραση των δεδομένων, τη μετακίνηση αυτών ή το μηχανισμό επικοινωνίας που χρησιμοποιείται στο φυσικό επίπεδο. Το επιπλέον κόστος διατήρησης της συνέπειας με λογισμικό και η μεγάλη καθυστέρηση στην επικοινωνία με μηνύματα καθιστούν το στόχο αυτό δύσκολα επιτεύξιμο.

System(year)	Developer	Implementation Method	Granularity	MC Model	CC Protocol
IVY (1986)	Yale	User-level library + OS modification, Apollo domain workstation, Aegis OS, 12Mbps token ring	Page (1KB)	SC	WI (Write Invalidate)
Munin (1991)	Rice	Runtime system library + Preprocessor + OS modification, Sun 3 workstation, System V, Ethernet	Variable size objects	Eager RC	Type-specific (WU/WI)
TreadMarks (1992)	Rice	User-level library, DECstation-5000/240, 100Mbps ATM or 10Mbps Ethernet.	Page (4KB)	LRC	WI
CVM (1997)	Maryland	User-level library, Solaris 2.5 on Sparcs, AIX4.1 on DEC Alpha	Page	LRC-MW, LRC-SW, SC	MW (Multiple Writer), WI
Midway (1991)	CMU	Runtime system + compiler, DECstation 5000, Mach OS 3.0, Ethernet LAN	Variable size objects	EC, PC, RC	WU (Write Update)
NCP2 (1997)	UFRJ,Brail	TreadMarks-like, PCI-based Hardware support, Cluster of PowerPC 604s, Myrinet Switch, LINUX	Page (4KB)	Affinity EC, RC	WU/WI
Quarks (1995)	Utah	User-level library, SunOS 4.1, HP-UX	Region, Page	RC, SC	WU/WI, MW
softFLASH (1996)	Stanford	OS modification, two level design, SGI Power Challenge, 100Mbyte/sec HIPPI,Irix 6.2.	Page(16k)	Epoch-based RC, DIRC	FLASH-like
Cashmere-2L (1997)	Rochester	Runtime library, Two level design, 8-node, DEC AlphaServer, Memory Channel	Page(8k)	Moderately HLRC	WU
Brazos (1997)	Rice	Runtime library +multithread, Windows NT. Cluster of Compaq Proliant 1500, 100Mbps Ethernet	Page	ScC	Early update, WU
Shasta (1996)	DEC WRL	Compiler support, Alpha Workstation, Memory Channel or ATM	variable granularities	SC	WI
Mermaid (1990)	Toronto	User-level library + OS modifications, SunOS	Page (1KB,8KB)	SC	WI
Dsoftware DSM6K (1993)	IBM Research	OS modification, IBM RISC System/6000, AIX v3,	Page (4KB)	SC	WI
Mirage (1989)	UCLA	OS Kernel, VAX 11/750, Ethernet	512 bytes	SC	WI
Plus (1990)	CMU	PLUS kernel, Motorola 8800, Caltech mesh routing	Page (4KB)	PC	WU
Simple-COMA (1995)	SICS(Sweden) and SUN	OS Modification and Hardware Support, Simulated 16-node DEC Alpha Multiprocessor,	Page alloc., fine-grain comm. & coh.	SC	WI
Blizzard-S (1994)	Wisconsin	User-level library + OS kernel modification	Cache line	SC	WI
Shrimp (1996)	Princeton	OS modification and hardware support, 16 Pentium PC nodes, Intel Paragon	Page	AURC, SC	WU/WI
Linda (1986)	Yale	Language, Ethernet based MicroVax and Intel iPSC hypercube	Variable size	SC	Impl. dependent
Orca (1988)	Vrije Univ., Netherlands	Language	Object size	Syn. dependent	WU

Πίνακας 2.1: Ορισμένες αντιπροσωπευτικές κατανεμημένες κοινές μνήμες λογισμικού και τα κυριότερα χαρακτηριστικά τους [28].

Τα συστήματα κατανεμημένης κοινής μνήμης υλοποιημένα με λογισμικό υποφέρουν από το υψηλό επιπλέον κόστος στο σύστημα που επιβάλλει η κλήση και η επεξεργασία του πρωτοκόλλου, από το μεγάλο καταμερισμό στην επικοινωνία και στη συνοχή (που δημιουργεί προβλήματα ψευδοδιαμοίρασης (false sharing) και επιπλέον επικοινωνιακές ανάγκες), καθώς και από υψηλές καθυστερήσεις στην επικοινωνία, σημείο ιδιαίτερα κρίσιμο για συστάδες υπολογιστών [28]. Ωστόσο χρησιμοποιούνται εξεζητημένα πρωτόκολλα για τη βελτίωση της επίδοσης των συστημάτων αυτών, που στοχεύουν κύρια στην προσπάθεια εξάλειψης του φαινομένου της ψευδο-διαμοίρασης, στη μείωση της απομακρυσμένης επικοινωνίας, στην απόκρυψη της καθυστέρησης στην επικοινωνία και στη μείωση του κόστους επεξεργασίας του πρωτοκόλλου.

Παρά τις παραπάνω δυσκολίες που έχουν προφανή αντίκτυπο στις επιδόσεις των συστημάτων κατανεμημένης κοινής μνήμης υλοποιημένων με λογισμικό υπάρχουν σαφή πλεονεκτήματα που τα καθιστούν ανταγωνιστικά: Κατ' αρχάς τα συστήματα αυτά περικλείουν όλα τα πλεονεκτήματα των συστημάτων κατανεμημένης κοινής μνήμης που ήδη αναφέραμε (εύκολη επέκταση, χαμηλό κόστος, προγραμματιστική ευκολία). Επιπρόσθετα, εφόσον δεν απαιτούν πολύπλοκες και ειδικού σκοπού διατάξεις υλικού, παρά υλοποιούν την ψευδαίσθηση της κοινής μνήμης με λογισμικό και ξεχωριστές μονάδες υλικού που επικοινωνούν με μηνύματα, προσφέρουν ακόμη χαμηλότερο κόστος και μεγαλύτερη επεκτασιμότητα.

2.4 Το σύστημα κατανεμημένης κοινής μνήμης υλοποιημένης με λογισμικό JIAJIA

2.4.1 Γενικά περί του JIAJIA

Διαλέξαμε να μελετήσουμε και να τροποποιήσουμε το JIAJIA [4][1] μία απλή και αποδοτική υλοποίηση με λογισμικό μιας κατανεμημένης κοινής μνήμης. Το JIAJIA αναπτύχθηκε κύρια από τον Weisong Shi στο Κέντρο Υπολογισμού Υψηλής Επίδοσης του Ινστιτούτου Τεχνολογίας Υπολογιστών της Κινέζικης Ακαδημίας Επιστημών. Προσφέρεται μαζί με τον πηγαίο κώδικα υπό μία άδεια χρήσης ανάλογη της παλαιού τύπου BSD. Τρέχει σε Solaris (SunOS), AIX, Linux και δοκιμάζεται και σε MS Windows NT. Αποτελείται από περίπου 5600 γραμμές κώδικα C, χωρισμένο σε 19 αρχεία. Η παρεχόμενη βιβλιογραφία περιλαμβάνει ένα εγχειρίδιο χρήσης, τεχνικές αναφορές, άρθρα, τη διδακτορική διατριβή του δημιουργού και φυσικά τον (ως επί το πλείστον χωρίς σχόλια) πηγαίο κώδικα. Η έκδοση με την οποία κύρια ασχοληθήκαμε ήταν η 2.2, η οποία έγινε διαθέσιμη τον Οκτώβριο του 1999.

2.4.2 Χαρακτηριστικά του JIAJIA

Στόχος των προγραμματιστών του JIAJIA ήταν η επέκταση του χώρου μνήμης και η βελτίωση της επίδοσης σε σχέση με υπάρχουσες κατανεμημένες κοινές μνήμες υλοποιημένες με λογισμικό. Στο JIAJIA ο κοινός χώρος μνήμης μπορεί να είναι μεγάλος όσο το σύνολο όλων των τοπικών μνημών, σε αντίθεση με άλλα συστήματα όπου το μέγεθος της κοινής μνήμης περιορίζεται από το μέγεθος της τοπικής μνήμης ενός κόμβου. Το JIAJIA προσφέρει ένα νέο μοντέλο

[1] JIAJIA στα Κινέζικα σημαίνει «το καλύτερο» και προφέρεται «τζα-τζα». [29]

διατήρησης της συνέπειας της μνήμης που χρησιμοποιεί πρωτόκολλο συνοχής και τήρηση σειράς διαδοχής γεγονότων για κάθε επεξεργαστή.

Το σχήμα οργάνωσης της μνήμης μοιάζει με αυτό της ανομοιόμορφης πρόσβασης (Non Uniform Memory Access — NUMA) και επιτρέπει απλή διαχείριση της κοινής μνήμης, δίχως ανάγκη για διατήρηση διαφορών (diffs), για συλλέκτη απορριμμάτων, για μετάφραση τοπικών διευθύνσεων σε καθολικές ή διατήρηση διανυσμάτων χρονοσήμων (timestamp vectors).

Το μοντέλο συνέπειας της μνήμης είναι συνέπεια εμβέλειας (scope consistency): Τα δεδομένα συσχετίζονται με αντικείμενα συγχρονισμού και στα σημεία συγχρονισμού μόνο τα δεδομένα που συσχετίζονται με το συγκεκριμένο αντικείμενο συγχρονισμού διατηρούνται συνεπή. Η συσχέτιση δεν προγραμματίζεται ρητά, αλλά επιτυγχάνεται δυναμικά όταν συμβαίνει μία αστοχία εγγραφής στη μνήμη εντός κάποιας εμβέλειας. Η συνέπεια εμβέλειας επιτυγχάνει τη μείωση του προβλήματος της ψευδοδιαμοίρασης, ενώ διατηρεί τη δυνατότητα προνοητικής μεταφοράς (prefetching) σελίδων και κρατά το επιπλέον λογιστικό κόστος περιγραφής της κατάστασης του συστήματος σχετικά χαμηλό. Η συνέπεια εμβέλειας του JIAJIA είναι αρκετά χαλαρή και απλούστερη στην υλοποίηση από τη συνέπεια αδρανούς απελευθέρωσης (Lazy Release Consistency — LRC). Το JIAJIA χρησιμοποιεί πρωτόκολλο εγγραφής - ακύρωσης (write - invalidate) και μάλιστα πολλαπλών εγγραφέων (multiple - writer), ώστε να μειώνεται το φαινόμενο της ψευδοδιαμοίρασης (false sharing).

Το πρωτόκολλο διατήρησης της συνοχής των λανθάνουσων μνημών είναι βασισμένο σε κλειδιά αμοιβαίου αποκλεισμού (lock - based). Δε διατηρείται κατάλογος, παρά οι ειδοποιήσεις εγγραφών (write notices) ενσωματώνονται στο εκάστοτε κλειδί.

Ένα ακόμη χαρακτηριστικό του JIAJIA είναι το γεγονός ότι η έννοια της έδρας (home) αναφέρεται όχι μόνο στα δεδομένα, αλλά και στις πληροφορίες συνοχής. Έτσι και τα μεν και τα δε έχουν κάποιο κόμβο στον οποίο ανήκουν.

Το JIAJIA χρησιμοποιεί τη μετανάστευση εργασιών (task migration), η οποία περιλαμβάνει τη μετακίνηση της υποεργασίας υπολογισμού μαζί με την αντίστοιχη υποεργασία δεδομένων. Ακόμη επιτρέπει την αλλαγή έδρας σελίδων.

Η μέθοδος που χρησιμοποιεί το JIAJIA για το χρονοπρογραμματισμό βρόχων είναι αυτοχρονοπρογραμματισμός βασισμένος σε συγγένειες (affinity-based self scheduling).

Έχει αναπτυχθεί τέλος ένα σχήμα βελτιστοποίησης της επικοινωνίας επιπέδου χρήστη ειδικά για συστήματα κατανεμημένων κοινών μνημών λογισμικού βασισμένων σε έδρες, τα οποία χρησιμοποιούν ως διασυνδετικό δίκτυο το Myrinet.

2.4.3 Αρχιτεκτονική του JIAJIA

Το JIAJIA ακολουθεί το προγραμματιστικό μοντέλο Μοναδικής Ροής Προγράμματος και Πολλαπλής Ροής Δεδομένων (Single Program Multiple Data — SPMD). Με άλλα λόγια το ίδιο πρόγραμμα εκτελείται σε διαφορετικά δεδομένα στους διάφορους κόμβους, χωρίς βέβαια να υπάρχει συγχρονισμός σε επίπεδο εντολής, λόγω της σχετικής ανεξαρτησίας των κόμβων.

Η κατανομή της κοινόχρηστης μνήμης γίνεται ως εξής: Κάθε μοιραζόμενη σελίδα έχει έναν κόμβο ως έδρα και οι έδρες μοιράζονται μεταξύ όλων των κόμβων. Σε έναν κόμβο οι αιτήσεις για εντός έδρας σελίδες εξυπηρετούνται τοπικά, ενώ οι αιτήσεις για εκτός έδρας σελίδες επιφέρουν τη μεταφορά των

σελίδων στον κόμβο και την τοπική αποθήκευσή τους σε λανθάνουσα μορφή. Οι αποθηκευμένες σελίδες μπορούν να βρίσκονται σε μία από τις ακόλουθες καταστάσεις (κατά τα γνωστά): Άκυρη (invalid), μόνο - ανάγνωσης (read-only) ή ανάγνωσης - εγγραφής (read-write). Οι γηρασμένες αποθηκευμένες σελίδες αντικαθίστανται εάν υπάρχει ανάγκη για χώρο και έτσι το JIAJIA υποστηρίζει κοινή μνήμη μεγαλύτερη από τη φυσική μνήμη μίας μόνο μηχανής. Στο JIAJIA μία κοινή σελίδα μένει πάντα και σε όλους τους σταθμούς στην ίδια διεύθυνση χρήστη, ανεξάρτητα από το αν βρίσκεται προσωρινά αποθηκευμένη στη λανθάνουσα μνήμη ή μόνιμα εντός έδρας. Κατά συνέπεια δεν απαιτείται μετάφραση διευθύνσεων μεταξύ διαφορετικών σταθμών.

Για το συγχρονισμό των διεργασιών το JIAJIA προσφέρει κλειδιά locks, φράγματα barriers και μεταβλητές υπό όρους (conditional variables (set, reset, wait until set)).

Το JIAJIA χρησιμοποιεί UDP/IP για την επικοινωνία μεταξύ των κόμβων, σε συνδυασμό με το δικό του πρωτόκολλο επικοινωνίας με βεβαιώσεις λήψης που εξασφαλίζει αξιόπιστη και σε σωστή σειρά μεταφορά.

2.4.4 Διεπαφή με τον προγραμματιστή εφαρμογών που χρησιμοποιούν το JIAJIA

Η διεπαφή με τον προγραμματιστή εφαρμογών (Application Programming Interface — API) που προσφέρει το JIAJIA έγκειται στην κλήση συναρτήσεων είτε για βασικές, είτε για προχωρημένες λειτουργίες [12]. Οι βασικές λειτουργίες συνίστανται στην αρχικοποίηση, την εκκίνηση δηλαδή της εφαρμογής σε όλους τους κόμβους, στην κατανομή κοινής μνήμης, στην απόκτηση και την απελευθέρωση κλειδιού και στην εκτέλεση καθολικού φράγματος. Οι προχωρημένες λειτουργίες συνίστανται στη ρύθμιση (της μετανάστευσης, του διανύσματος εγγραφών ή της καθολικής μετάδοσης των μηνυμάτων φράγματος),

στην εξισορρόπηση φορτίου (διαίρεση εργασιών ή έλεγχος φορτίου), στη χρήση μεταβλητών υπό όρους για συγχρονισμό (μαζί με κλειδιά για συνοχή), στην εκτέλεση αναμονής (αντίστοιχη του φράγματος, αλλά χωρίς λειτουργίες σχετιζόμενες με τη συνοχή), στη συλλογή στατιστικών στοιχείων ή χρόνων και σε κλήσεις που μοιάζουν με τη διεπαφή περάσματος μηνυμάτων MPI (Message Passing Interface [10]) (αποστολή, λήψη, καθολική αποστολή, μείωση (μετακίνηση πληροφορίας μεταξύ των αποθηκευτικών χώρων των κόμβων».

2.4.5 Επιλογές ρυθμίσεων του JIAJIA

Μετανάστευση (Home migration)

Το JIAJIA προσφέρει προαιρετικά ένα στοιχειώδη μηχανισμό για την προσαρμοστική αλλαγή έδρας των σελίδων μνήμης, τον οποίο και βελτιώσαμε. Σύμφωνα με αυτόν, οι σελίδες που εγγράφονται μόνο από έναν επεξεργαστή μεταξύ δύο φραγμάτων μεταναστεύουν στο μοναδικό αυτό εγγραφέα. Ο μηχανισμός αυτός ενεργοποιείται μετά την αρχικοποίηση των κοινών δεδομένων, ώστε ο αρχικοποιητής να μη γίνει έδρα όλων των σελίδων. Με τον παραπάνω μηχανισμό μειώνεται η μετακίνηση διαφορών και οι επιδόσεις βελτιώνονται. Επιπλέον εξαλείφεται το φαινόμενο της προαποστολής δεδομένων, με την έννοια ότι δεν υπάρχει η ανάγκη για έναν εγγραφέα που δεν είναι έδρα να στείλει τις διαφορές μίας τροποποιημένης σελίδας στην έδρα της, εάν αυτός πρόκειται να γίνει η νέα έδρα. Ο παραπάνω απλός μηχανισμός λειτουργεί καλά για κανονικά προβλήματα με μεγάλες ομάδες διαμοιραζόμενων δεδομένων και με φράγματα ως κύρια μέθοδο συγχρονισμού.

Διάνυσμα εγγραφών (Write vector)

Στο JIAJIA μπορεί να χρησιμοποιηθεί ένα διάνυσμα εγγραφών για να μειώσει το μέγεθος των μηνυμάτων που είναι αναγκασμένοι να ανταλλάσσουν οι κόμβοι. Με βάση αυτό, αντί να μεταφέρεται ολόκληρη η σελίδα όταν συμβεί ένα σφάλμα σελίδας, μεταφέρονται μόνο τα τμήματα της που έχουν τροποποιηθεί σε σχέση με την παλαιότερα μεταφερμένη σελίδα. Η παραπάνω πρακτική λειτουργεί καλά για ακανόνιστα προβλήματα με μικρές ομάδες διαμοιραζόμενων δεδομένων και κοινόχρηστες σελίδες στις οποίες γίνονται συχνά αναφορές. Το JIAJIA προσφέρει επίσης ένα προσαρμοστικό σχήμα ανίχνευσης εγγραφών που μειώνει τα σφάλματα εγγραφών σε σελίδες που προορίζονται μόνο για ανάγνωση.

Καθολική μετάδοση των μηνυμάτων φράγματος (Broadcast barrier messages)

Σύμφωνα με την καθολική μετάδοση μηνυμάτων φράγματος που προσφέρει το JIAJIA, αφού όλοι οι κόμβοι φθάσουν σε ένα φράγμα ο διαχειριστής στέλνει βεβαίωση λήψης σε όλους, είτε σε έναν προς έναν, είτε σε δενδροειδή δομή. Για δίκτυα με δομή διαύλου όπως το Ethernet προτείνεται η αποστολή σε έναν προς ένα, αφού η δενδροειδής δομή εισάγει επιπλέον κόστος.

Εξισορρόπηση φορτίου (Load balancing)

Η υποστήριξη εξισορρόπησης φορτίου που προσφέρει το JIAJIA είναι ακόμη απλή. Περιλαμβάνει τη διαίρεση των εργασιών (βρόχων) σε όλους τους επεξεργαστές σύμφωνα με την υπολογιστική ισχύ αυτών αντί μιας ίσης κατανομής.

Επιπλέον περιλαμβάνεται έλεγχος και καταγραφή της υπολογιστικής ισχύς κάθε επεξεργαστή του συστήματος (υπολογιζόμενης μέσω του καταναλωθέντα υπολογιστικού χρόνου).

Κεφάλαιο 3

Λογική της λύσης και υλοποίηση

3.1 Σημασία της προώθησης σελίδων μνήμης στις συστάδες υπολογιστών

3.1.1 Αναγκαιότητα της προώθησης σελίδων μνήμης στις συστάδες υπολογιστών

Τα βασικά χαρακτηριστικά των μετα-υπολογιστικών περιβαλλόντων (meta - computing environments), δηλαδή η ετερογένεια, η μεταβλητότητα των πόρων, η κατανομή και η μη αποκλειστική χρήση μειώνουν γενικά την απόδοση των εφαρμογών και καθιστούν τα συστήματα αυτά συνήθως λιγότερο αποτελεσματικά από τους πολυεπεξεργαστές [35]. Προκειμένου ένα τέτοιο σύστημα —στην προκειμένη περίπτωση μια συστάδα υπολογιστών— να μεγιστοποιεί την απόδοσή του, θα πρέπει να μπορεί να προσαρμόζεται τόσο στους διαθέσιμους υπολογιστικούς πόρους, όσο και στις εγγενείς υπολογιστικές και επικοινωνιακές απαιτήσεις της εκάστοτε εφαρμογής [27].

Η προσαρμογή στους διαθέσιμους υπολογιστικούς πόρους —ώστε να εξασφαλίζεται υψηλή εκμετάλλευση αυτών— μπορεί να επιτευχθεί με τη *δυναμική μετανάστευση εργασιών (dynamic task migration)*. Με βάση τη λεπτότητα του καταμερισμού (granularity) μπορούμε να χωρίσουμε τις μέχρι τώρα προσπάθειες στον τομέα αυτό σε τρεις κατηγορίες [28]:

- Επιπέδου βρόχου (loop-level), όπου έχουμε μετακίνηση βρόχων.
- Επιπεδου νήματος (thread-level), όπου έχουμε μετανάστευση νημάτων.
- Επιπέδου διεργασίας (process-level), όπου έχουμε μετανάστευση διεργασιών.

Τα μεταναστευτικά σχήματα επιπέδου βρόχου και νήματος χρησιμοποιούνται ευρέως σε περιβάλλοντα κοινής μνήμης, όπως οι πολυεπεξεργαστές κοινής μνήμης και τα συστήματα κατανεμημένης κοινής μνήμης λογισμικού. Η μετανάστευση εργασιών επιπέδου διεργασίας υιοθετείται σε περιβάλλοντα κοινής μνήμης, αλλά πιο συχνά σε περιβάλλοντα περάσματος μηνυμάτων, όπως τα κατανεμημένα συστήματα (όπου οι δύο πρώτοι τύποι μετακινήσεων δεν αποδίδουν συνήθως ιδιαίτερα καλά) και οι μαζικά παράλληλοι επεξεργαστές (massively parallel processors — MPPs).

Η συνήθης αντιμετώπιση του προβλήματος της μετανάστευσης έργου (ή αλλιώς εργασιών — (tasks)), τόσο σε επίπεδο βρόχου, όσο και σε επίπεδο νήματος έχει ωστόσο το εξής μειονέκτημα: Αφορά μόνο το τμήμα του κώδικα που εκτελεί κάποιον υπολογισμό, αγνοώντας τα δεδομένα που αναφέρονται στον υπολογισμό αυτό. Έτσι όταν μια εργασία μεταναστεύει από έναν επεξεργαστή σε έναν άλλο τα δεδομένα που αυτή χρησιμοποιεί μένουν στον προηγούμενο επεξεργαστή. Κατά συνέπεια ο νέος επεξεργαστής είναι αναγκασμένος

να προσπελαύνει απομακρυσμένα δεδομένα όταν εκτελεί την εργασία και προφανώς χάνεται το όφελος της μετανάστευσης της εργασίας, η και χειροτερεύει ακόμη η απόδοση. Έτσι είναι σκόπιμο να επαναπροσδιορίσουμε το έργο (task), ως άθροισμα δύο υποέργων (subtasks), τα οποία θα μεταναστεύουν μαζί: του υποέργου υπολογισμού και του υποέργου δεδομένων [28]. Το υποέργο υπολογισμού αναφέρεται στο τμήμα του προς εκτέλεση κώδικα, ενώ το υποέργο δεδομένων αναφέρεται στις λειτουργίες πρόσβασης των δεδομένων που βρίσκονται στη μνήμη. Συγκεκριμένα για τα συστήματα κατανεμημένης κοινής μνήμης, στα οποία οι υπολογισμοί γίνονται πολύ πιο γρήγορα από τις προσβάσεις στη μνήμη, η σημασία του υποέργου δεδομένων είναι ιδιαίτερα αυξημένη. Το JIAJIA προσφέρει δυναμική μετανάστευση εργασιών επιπέδου βρόχου μέσω της ενεργοποίησης της εξισορρόπησης φορτίου. Λαμβάνει πάντως υπ΄ όψιν του στη μετανάστευση και το υποέργο δεδομένων, αφού οι αποκλειστικά εγγραφόμενες σελίδες, οι οποίες είναι κι εκείνες που συμμετείχαν στον υπολογισμό, μεταναστεύουν σε εκείνους τους σταθμούς που τις χρησιμοποίησαν.

Από τα παραπάνω συνάγεται ότι είναι επιθυμητό να τακτοποιούμε έτσι διεργασίες (ή νήματα σε ένα πολυνηματικό περιβάλλον) και σελίδες μνήμης, ώστε κάθε διεργασία (ή νήμα) να βρίσκεται στον ίδιο κόμβο με τις σελίδες που προσπελαύνει πιο συχνά [20]. Το σύνολο αυτών των σελίδων το ονομάζουμε *σύνολο συγγένειας μνήμης (memory affinity set)* μιας διεργασίας (ή ενός νήματος). Η διατήρηση της συγγένειας διεργασιών (ή νημάτων) και μνήμης είναι σημαντική, ώστε να αποφεύγονται οι «ορφανές» σελίδες και οι απομακρυσμένες προσβάσεις μνήμης [24].

Η παραπάνω απαίτηση όμως είναι σημαντική ακόμη και για περιπτώσεις που δε μας απασχολεί η μετανάστευση εργασιών και η βελτίωσή της. Πρακτικά, οι σελίδες μνήμης τοποθετούνται συνήθως είτε στον κόμβο που θα τις ζητήσει πρώτος, είτε στον κόμβο που καθορίζει η εφαρμογή ρητά μέσα από την κλήση των συναρτήσεων κατανομής μνήμης. Στην περίπτωση που κάθε σελίδα δεν

έχει μια συγκεκριμένη, σταθερή έδρα (home) το σύστημα είναι πιο ευπροσάρμοστο στις υπολογιστικές και επικοινωνιακές απαιτήσεις των εφαρμογών, οι οποίες μπορεί μάλιστα να μεταβάλλονται κατά τη διάρκεια της εκτέλεσής τους. Προβάλλει συνεπώς η ανάγκη για μετανάστευση δεδομένων, άρα και για *δυναμική μετανάστευση σελίδων μνήμης (dynamic page migration)*, ή αλλιώς για *προώθηση σελίδων μνήμης (page forwarding)* στους σταθμούς μιας συστάδας υπολογιστών που τις χρησιμοποιούν περισσότερο κάθε στιγμή, ώστε να μειώνονται οι απομακρυσμένες και αργές λόγω διασυνδετικού δικτύου προσβάσεις μνήμης. Η δυναμική μετανάστευση σελίδων εντείνει την ακρίβεια, την επικαιρότητα, την ευελιξία, την ετοιμότητα απόκρισης σε αλλαγές κατά την εκτέλεση και την προσαρμοστικότητα των συστημάτων [23].

Ακόμη και σε ένα αφοσιωμένο περιβάλλον η μετανάστευση σελίδων μπορεί να βελτιώσει την τοπικότητα των προσβάσεων στη μνήμη του συστήματος. Κατά συνέπεια μπορεί να μειώσει και σε ένα τέτοιο περιβάλλον σημαντικά την παραγωγή διαφορών και διδύμων για τη δημιουργία αυτών, μιας και οι εγγραφές σε τοπικές (εντός έδρας) σελίδες δεν παράγουν τίποτα από τα παραπάνω σε συστήματα βασισμένα σε έδρες. Προφανώς βελτιώνεται έτσι η απόδοση του συστήματος.

3.1.2 Σημασία της προώθησης σελίδων μνήμης για πρωτόκολλα βασισμένα σε έδρες

Οι κατανεμημένες κοινές μνήμες λογισμικού στις οποίες κάθε κοινή σελίδα έχει έναν συγκεκριμένο σταθμό ως έδρα αποτελούν μία απλή, αποτελεσματική και κλιμακώσιμη λύση [14]. Σε ένα πρωτόκολλο βασισμένο σε έδρες, κάθε κοινή σελίδα έχει μία προσδιορισμένη έδρα, στην οποία εκπέμπονται όλες οι εγγραφές της και από την οποία παράγονται όλα τα αντίγραφά της. Σε ένα πρωτόκολλο πολλαπλών εγγραφέων που υλοποιεί ένα χαλαρό μοντέλο συνέπειας μνήμης,

οι διαφορές που δημιουργήθηκαν σε ένα διάστημα συγχρονισμού στέλνονται στις αντίστοιχες έδρες στο τέλος του διαστήματος αυτού. Τα εκτός έδρας αντίγραφα ακυρώνονται στην επόμενη λειτουργία απόκτησης κλειδιού αμοιβαίου αποκλεισμού, σύμφωνα με τις απαιτήσεις του μοντέλου συνέπειας μνήμης. Συγκρινόμενο με ένα σχήμα οργάνωσης μνήμης χωρίς έδρες (όπως για παράδειγμα αυτό της συνέπειας αδρανούς απελευθέρωσης), το βασισμένο σε έδρες σχήμα έχει μερικά βασικά πλεονεκτήματα:

- Όταν συμβεί σφάλμα σελίδας, απαιτείται μόνο ένα ταξίδι με επιστροφή μέχρι την έδρα μιας σελίδας για να αποκτήσει κάποιος σταθμός αντίγραφό της.
- Οι εγγραφές σε σελίδες εντός έδρας δεν παράγουν δίδυμες σελίδες και διαφορές.
- Οι διαφορές εφαρμόζονται στην έδρα μία φορά μαζικά και η επίδραση της μόλυνσης της λανθάνουσας μνήμης είναι αμελητέα.
- Οι διαφορές στέλνονται στην έδρα πρόθυμα στο επόμενο σημείο συγχρονισμού και έτσι καταναλώνεται πολλή λιγότερη μνήμη και εξαλείφεται η ανάγκη για συλλογή απορριμμάτων.
- Δεν υπάρχει πρόβλημα συσσώρευσης διαφορών.
- Ένα εκτός έδρας αντίγραφο μιας σελίδας μπορεί να αντικατασταθεί και οι διαφορές να εφαρμοσθούν στο εντός έδρας αντίγραφο, αν η τοπική μνήμη δεν επαρκεί πλέον, ενώ σε άλλες αρχιτεκτονικές πρέπει να υπάρχει μηχανισμός εξασφάλισης ότι το τελευταίο αντίγραφο μιας σελίδας δε θα αντικατασταθεί.
- Ο αλγόριθμος αντικατάστασης σελίδων της λανθάνουσας μνήμης είναι εύκολος στην υλοποίηση. Η κατάσταση συνοχής είναι απλή.

Από την άλλη, μερικά εν δυνάμει μειονεκτήματα του βασισμένου σε έδρες σχήματος είναι και τα εξής:

- Όλη η σελίδα μεταφέρεται σε περίπτωση που συμβεί σφάλμα σελίδας.
- Οι επιδόσεις εξαρτώνται σε μεγάλο βαθμό από την κατανομή των εδρών.

Το τελευταίο χαρακτηριστικό των βασισμένων σε έδρες κατανεμημένων κοινών μνημών λογισμικού είναι ιδιαίτερα σημαντικό. Καλές επιδόσεις αναμένονται μόνο όταν οι έδρες των σελίδων είναι καλά διαμοιρασμένες, ώστε οι περισσότερες προσβάσεις κάθε επεξεργαστή να γίνονται τελικά εντός έδρας. Ειδάλλως η παραγωγή και εφαρμογή διαφορών ανυπόμονα κατά την απελευθέρωση κλειδιών αμοιβαίου αποκλεισμού ίσως εισάγει πολύ περισσότερο επιπλέον κόστος από την παραγωγή και την εφαρμογή αυτών αργότερα, όταν συμβούν σφάλματα σελίδας, όπως συμβαίνει στη συνέπεια αδρανούς απελευθέρωσης. Συνεπώς η δυνατότητα δυναμικής μετανάστευσης σελίδων επιτρέπει την αναπροσαρμογή της κατανομής των εδρών, ώστε κάθε στιγμή και για κάθε εφαρμογή να ελαχιστοποιούνται οι απομακρυσμένες προσβάσεις μνήμης και να αυξάνεται η απόδοση.

Η δυνατότητα της ενεργοποίησης της δυναμικής αναπροσαρμογής της κατανομής των εδρών δε μειώνει βέβαια τη σημασία του σαφούς ελέγχου της αρχικής κατανομής των κοινών δεδομένων. Μία σωστή αρχική κατανομή των δεδομένων σε έδρες μπορεί να βοηθήσει στην αποφυγή περιττών αρχικών μεταναστεύσεων. Επομένως μία κατανεμημένη κοινή μνήμη λογισμικού βασισμένη σε έδρες είναι χρήσιμο να προσφέρει τη δυνατότητα ρητού ορισμού των εδρών κατά την αρχική κατανομή των δεδομένων.

3.1.3 Τοπικότητα στις κατανεμημένες κοινές μνήμες

Η τοπικότητα (Locality) των δεδομένων είναι ιδιαίτερης σημασίας σε κάθε υλοποίηση κατανεμημένης κοινής μνήμης, αφού οι απομακρυσμένες προσβάσεις στη μνήμη απαιτούν πολύ περισσότερο χρόνο από τις τοπικές. Επιπλέον στις κατανεμημένες κοινές μνήμες λογισμικού η τοπικότητα είναι ακόμη πιο σημαντική, αφού η καθυστέρηση των διαδιεργασιακών κλήσεων είναι μεγάλη, ενώ και η εξυπηρέτηση απομακρυσμένων προσβάσεων κοστίζει κύκλους του επεξεργαστή. Έτσι είναι προφανής η σημασία της εκμετάλλευσης της τοπικότητας για τη βελτίωση των επιδόσεων στην εκτέλεση παράλληλων εφαρμογών που χρησιμοποιούν το JIAJIA [12]. Μιλώντας για τοπικότητα στην περίπτωση του JIAJIA αναφερόμαστε τόσο στην τοπικότητα της λανθάνουσας μνήμης (δηλαδή χωρική και χρονική επιτυχία στην ανεύρεση δεδομένων από αυτή), όσο και στην τοπικότητα της έδρας (δηλαδή χωρική και χρονική επιτυχία στην ανεύρεση δεδομένων από τον κόμβο που τα αναζητά). Προσπαθώντας να βελτιώσουμε την τοπικότητα μπορούμε — εκτός του σαφούς ελέγχου της κατανομής των δεδομένων μεταξύ των επεξεργαστών κατά την αρχική δέσμευση κοινής μνήμης — να ρυθμίσουμε από τον πηγαίο κώδικα του JIAJIA το μέγεθος της λανθάνουσας μνήμης ή και το μέγεθος των σελίδων μνήμης. Φυσικά η δυνατότητα μετανάστευσης σελίδων επιτρέπει τη δυναμική αναπροσαρμογή του συστήματος, για τη μεγιστοποίηση της εκμετάλλευσης της τοπικότητας.

3.2 Ο υπάρχων μηχανισμός μετανάστευσης σελίδων

3.2.1 Λογική του υπάρχοντος μηχανισμού μετανάστευσης σελίδων

Όπως ήδη αναφέρθηκε, η επίδοση των κατανεμημένων κοινών μνημών λογισμικού όπου κάθε σελίδα έχει συγκεκριμένη έδρα εξαρτάται κατά πολύ από την κατανομή των εδρών των σελίδων. Μια καλή κατανομή των εδρών των σελίδων θα πρέπει να κρατά την έδρα κάθε κοινής σελίδας στον επεξεργαστή που την γράφει πιο συχνά, ώστε αυτός να μπορεί να τη γράφει άμεσα (αφού είναι η έδρα της) και να μειώνονται οι *διαφορές (diffs)* που πρέπει να παραχθούν και να εφαρμοσθούν.

Ένα πρόβλημα που ανακύπτει κατά τη μετανάστευση σελίδων είναι ότι πρέπει να υπάρχει μηχανισμός ενημέρωσης όλων των σταθμών για τη νέα έδρα μιας σελίδας που μετανάστευσε. Μια απλή αντιμετώπιση του προβλήματος θα ήταν η εκπομπή προς όλους τους σταθμούς (broadcast) της νέας έδρας κάποιας σελίδας. Η καθολική εκπομπή ωστόσο είναι πολύ ακριβή, μιας και δημιουργεί μεγάλη κίνηση στο διασυνδετικό δίκτυο. Προκειμένου να μειωθεί λοιπόν το επιπλέον επικοινωνιακό κόστος, στο JIAJIA η απόφαση για τη μετανάστευση κάποιας σελίδας λαμβάνεται ξεχωριστά από την εκάστοτε έδρα, αλλά μεταδίδεται από το διαχειριστή κάθε φράγματος ως εξής: Η πληροφορία της μετανάστευσης *φορτώνεται (piggybacked)* στο μήνυμα παραχώρησης του φράγματος (barrier grant).

Προκειμένου να μειώσει ακόμη περισσότερο το επιπλέον κόστος που προκύπτει από τις μεταναστεύσεις, το JIAJIA αλλάζει την έδρα μιας σελίδας σε ένα φράγμα, μόνο όταν ο σταθμός που πρόκειται να γίνει η νέα έδρα ήταν ο μοναδικός

εγγραφέας της σελίδας κατά το τελευταίο διάστημα μεταξύ φραγμάτων. Με άλλα λόγια οι σελίδες που εγγράφονται μόνο από έναν επεξεργαστή μεταξύ δύο φραγμάτων μεταναστεύουν στο μοναδικό αυτό εγγραφέα. Στο πρωτόκολλο αυτό το αποθηκευμένο αντίγραφο μιας σελίδας ακυρώνεται σε ένα φράγμα, αν η σελίδα αυτή έχει μεταβληθεί κατά το διάστημα που μεσολάβησε από το προηγούμενο φράγμα. Ωστόσο, αν η σελίδα έχει μεταβληθεί μόνο από έναν επεξεργαστή στο τελευταίο διάστημα μεταξύ φραγμάτων και ο επεξεργαστής αυτός δεν είναι η έδρα της σελίδας, τότε είναι περιττό για αυτόν να ακυρώσει τη σελίδα από τη λανθάνουσα μνήμη του, μιας και η σελίδα αυτή είναι σωστή (αντίγραφα της σελίδας αυτής σε άλλους επεξεργαστές ακυρώνονται κανονικά). Κατά συνέπεια, εάν μια σελίδα μεταναστεύει στο μοναδικό εγγραφέα, δεν απαιτείται η μετάδοσή της, αφού η νέα έδρα έχει ήδη την πιο ενημερωμένη έκδοσή της. Το JIAJIA ενημερώνει όλους τους σταθμούς για τη μετανάστευση της σελίδας στο μήνυμα παραχώρησης του φράγματος και κάθε σταθμός αναγνωρίζει το μοναδικό εγγραφέα μιας σελίδας.

Μιας και κατά τη μετανάστευση μιας σελίδας δεν έχουμε μετάδοσή της, ο διαχειριστής του φράγματος πρέπει να εξασφαλίζει ότι η λανθάνουσα μνήμη της νέας έδρας έχει ένα αντίγραφο της σελίδας προς μετανάστευση. Αυτό δεν είναι απαραίτητο να συμβαίνει στο JIAJIA, διότι το αντίγραφο αυτό μπορεί να έχει αντικατασταθεί ήδη από κάποια άλλη σελίδα στη λανθάνουσα μνήμη του σταθμού αυτού. Το JIAJIA λύνει το πρόβλημα αυτό με μία ετικέτα (tag) σε κάθε *ειδοποίηση εγγραφής (write notice)* που στέλνεται στο διαχειριστή του φράγματος από το σταθμό που φθάνει στο φράγμα. Η ετικέτα αυτή δείχνει εάν η λανθάνουσα μνήμη του σταθμού που φθάνει στο φράγμα έχει έγκυρο αντίγραφο της σελίδας που δηλώνει ότι έχει γράψει.

3.2.2 Περιγραφή του υπάρχοντος αλγορίθμου μετανάστευσης σελίδων

Με βάση την παραπάνω ανάλυση, ο αλγόριθμος μετανάστευσης του JIAJIA μπορεί να περιγραφεί ως ακολούθως [14]:

1. Όταν ένας σταθμός φθάνει σε ένα φράγμα, στέλνει τις ειδοποιήσεις εγγραφής του διαστήματος που μεσολάβησε από το τελευταίο φράγμα στο διαχειριστή του φράγματος. Κάθε ειδοποίηση εγγραφής είναι η διεύθυνση μιας σελίδας που μεταβλήθηκε από το τελευταίο φράγμα. Το λιγότερο σημαντικό bit της ειδοποίησης εγγραφής λειτουργεί ως ετικέτα και παίρνει την τιμή «1» εάν η αντίστοιχη σελίδα είναι έγκυρη στη λανθάνουσα μνήμη.

2. Όταν ο διαχειριστής του φράγματος λαμβάνει μία αίτηση φράγματος (barrier request), καταγράφει τις ειδοποιήσεις εγγραφής που αυτή μεταφέρει. Κάθε ειδοποίηση εγγραφής συνδέεται με ένα πεδίο τροποποιητή (modifier) που δηλώνει τον αναγνωριστή του σταθμού που τροποποίησε τη σελίδα. Εάν μια σελίδα τροποποιήθηκε από περισσότερους από δύο σταθμούς, το πεδίο του τροποποιητή παίρνει την τιμή «πολλαπλοί» (multiple).

3. Όταν ληφθούν όλες οι αιτήσεις για ένα φράγμα, ο διαχειριστής του φράγματος αυτού εκπέμπει προς όλους τους σταθμούς όλες τις ειδοποιήσεις εγγραφών (μαζί με τα αντίστοιχο πεδίο τροποποιητή για κάθε εγγραφή).

4. Όταν ένας σταθμός λάβει μια απόκριση φράγματος (barrier reply), ελέγχει κάθε ληφθείσα ειδοποίηση εγγραφής και το αντίστοιχο πεδίο τροποποιητή. Εάν η ειδοποίηση εγγραφής και το αντίστοιχο πεδίο τροποποιητή δείχνουν πως η σελίδα τροποποιήθηκε κατά τη διάρκεια του τελευταίου διαστήματος μεταξύ φραγμάτων από πολλαπλούς σταθμούς ή από ένα σταθμό διαφορετικό από τον παρόντα, τότε η αντίστοιχη σελίδα ακυρώνεται εάν φυλάσσονταν στη λανθάνουσα μνήμη. Εάν η ειδοποίηση εγγραφής και

το αντίστοιχο πεδίο τροποποιητή δείχνουν πως η σελίδα τροποποιήθηκε μόνο από ένα σταθμό και πως η λανθάνουσα μνήμη του σταθμού αυτού έχει ένα έγκυρο αντίγραφο της σελίδας (το λιγότερο σημαντικό bit της ειδοποίησης εγγραφής είναι «1»), τότε η έδρα της σελίδας αλλάζει και γίνεται ο σταθμός που την τροποποίησε, ως εξής:

(α′) Εάν ο σταθμός είναι η νέα έδρα της μετακινηθείσας σελίδας, τότε δίνεται μια θέση για τη σελίδα αυτή στον πίνακα των σελίδων που εδράζονται στο σταθμό αυτό, η θέση της στον πίνακα των σελίδων που κρατούνται στη λανθάνουσα μνήμη αποδεσμεύεται και το πεδίο που αναφέρει το σταθμό-έδρα της σελίδας αυτής στην αντίστοιχη θέση του καθολικού πίνακα σελίδων τίθεται να δείχνει στη νέα έδρα.

(β′) Εάν ο σταθμός είναι η παλαιά έδρα της μετακινηθείσας σελίδας, τότε η αντίστοιχη θέση της στον πίνακα των σελίδων που εδράζονται στο σταθμό αυτό αποδεσμεύεται, η σελίδα ακυρώνεται εάν δεν υπάρχει ελεύθερη θέση στον πίνακα των σελίδων που κρατούνται στη λανθάνουσα μνήμη ή κρατείται ως έγκυρη εάν βρεθεί ελεύθερη θέση για τη σελίδα αυτή στον πίνακα αυτό και το πεδίο που αναφέρει το σταθμό-έδρα της σελίδας αυτής στην αντίστοιχη θέση του καθολικού πίνακα σελίδων τίθεται να δείχνει στη νέα έδρα.

(γ′) Εάν ο σταθμός δεν είναι ούτε η νέα ούτε η παλαιά έδρα της μετακινηθείσας σελίδας, τότε μόνο το πεδίο που αναφέρει το σταθμό-έδρα της σελίδας αυτής στην αντίστοιχη θέση του καθολικού πίνακα σελίδων τίθεται να δείχνει στη νέα έδρα.

Εύκολα μπορεί να διαπιστωθεί πως ο παραπάνω αλγόριθμος —προς όφελός του— δεν απαιτεί επιπλέον μηνύματα για τη μετανάστευση: Δεν απαιτείται ούτε η μετάδοση της σελίδας που μεταναστεύει, ούτε μηνύματα για την πληροφόρηση των σταθμών για τη μετανάστευση.

3.3 Ο νέος μηχανισμός μετανάστευσης σελίδων

3.3.1 Λογική του νέου μηχανισμού μετανάστευσης σελίδων

Το πρωτόκολλο μετανάστευσης σελίδων που ήδη ενσωματώνει το JIAJIA είναι αρκετά συντηρητικό, προκειμένου να αποφύγει και επιπλέον επικοινωνιακή επιβάρυνση. Μόνο οι σελίδες που εγγράφονται από έναν επεξεργαστή μεταξύ δύο φραγμάτων μεταναστεύουν στο μοναδικό αυτό εγγραφέα. Υπάρχουν όμως γενικά και σελίδες που, αν και εγγράφονται από περισσότερους του ενός επεξεργαστές, μεταβάλλονται ισχυρότερα από έναν από όλους, δηλαδή κάποιος από όλους τους επεξεργαστές εφαρμόζει περισσότερες διαφορές σε αυτές. Σε κάποιες περιπτώσεις μπορεί δε ο ισχυρότερος τροποποιητής μιας σελίδας να μην είναι αυτός που αποτελεί την έδρα της. Η μετανάστευση αυτών των σελίδων στον ισχυρό τροποποιητή —εφόσον οι διαφορές που εφαρμόζει στις σελίδες ξεπερνούν κάποιο κατώφλι— θα μπορούσε να βελτιώσει την τοπικότητα του συστήματος και άρα να μειώσει τις χρονοβόρες προσβάσεις σε απομακρυσμένες μνήμες και τις ακριβές δημιουργίες διαφορών, αυξάνοντας κατά συνέπεια τις επιδόσεις του συστήματος. Προφανώς όμως για να επιτευχθεί κάτι τέτοιο απαιτείται ένα πιο πολύπλοκο πρωτόκολλο, το οποίο μάλιστα θα πρέπει να αντιμετωπίζει και καταστάσεις που με το προηγούμενο πρωτόκολλο δεν εμφανίζονταν.

3.3.2 Περιγραφή του νέου αλγορίθμου μετανάστευσης σελίδων

Ο γενικευμένος αυτός αλγόριθμος μετανάστευσης σελίδων θα πρέπει να περιλαμβάνει τα εξής βασικά βήματα:

1. Κάθε σταθμός μετράει για κάθε σελίδα που εδράζεται σε αυτόν τα bytes των διαφορών που εφαρμόζονται σε αυτή από όλους τους σταθμούς. Αφού όλες οι διαφορές επιστρέφουν στην έδρα της σελίδας για να εφαρμοσθούν, η έδρα είναι η πλέον κατάλληλη για να μετράει τα bytes των διαφορών.

2. Όταν ένας σταθμός φθάσει σε ένα φράγμα στέλνει τις ειδοποιήσεις εγγραφής του διαστήματος που μεσολάβησε από το τελευταίο φράγμα στο διαχειριστή του φράγματος. Κατόπιν υπολογίζει —για κάθε σελίδα που εδράζεται σε αυτόν— το μέγεθος των μεγαλύτερων τροποποιήσεων και από ποιο σταθμό προήλθαν. Στη συνέχεια στέλνει (επίσης στο διαχειριστή του φράγματος) την αίτηση φράγματος, στην οποία φορτώνονται και οι διευθύνσεις των σελίδων που εδράζονται στο σταθμό και που τροποποιήθηκαν μαζί με τον αναγνωριστή του πιο ισχυρού τροποποιητή τους. Προκειμένου να αποφευχθούν συνεχείς μεταναστεύσεις σελίδων τίθεται κάποιο κατώφλι μεγέθους τροποποιήσεων, ώστε να μη φορτώνονται οι διευθύνσεις όλων των τροποποιημένων σελίδων, αλλά μόνο όσων έχουν τροποποιηθεί παραπάνω (πιο ισχυρά) από ένα ελάχιστο όριο (κατώφλι). Το κατώφλι αυτό εξαρτάται από την εφαρμογή, το διασυνδετικό δίκτυο και την επεξεργαστική ισχύ των σταθμών.

3. Όταν ο διαχειριστής του φράγματος λαμβάνει μία αίτηση φράγματος, καταγράφει τις ειδοποιήσεις εγγραφής που αυτή μεταφέρει, αλλά και τις διευθύνσεις των σελίδων που έχουν τροποποιηθεί (πάνω από ένα κατώφλι), καθώς και τον αναγνωριστή του πιο ισχυρού τροποποιητή τους. Έτσι ο διαχειριστής φράγματος συγκεντρώνει όλες τις ειδοποιήσεις εγγραφής, αλλά και τις διευθύνσεις όλων των (ισχυρά) τροποποιημένων σελίδων και τους τροποποιητές τους, ανεξάρτητα από το που εδράζονται οι σελίδες αυτές.

4. Όταν ληφθούν όλες οι αιτήσεις για ένα φράγμα, ο διαχειριστής του φράγματος αυτού εκπέμπει προς όλους τους σταθμούς όλες τις ειδοποιήσεις

εγγραφών, αλλά και όλες τις διευθύνσεις των (ισχυρά) τροποποιημένων σελίδων, μαζί με τους αντίστοιχους τροποποιητές. Οι πληροφορίες αυτές φορτώνονται στην απόκριση φράγματος.

5. Όταν ένας σταθμός λάβει μια απόκριση φράγματος, ελέγχει κάθε ληφθείσα ειδοποίηση εγγραφής και το αντίστοιχο πεδίο τροποποιητή. Εάν η ειδοποίηση εγγραφής και το αντίστοιχο πεδίο τροποποιητή δείχνουν πως η σελίδα τροποποιήθηκε κατά τη διάρκεια του τελευταίου διαστήματος μεταξύ φραγμάτων από πολλαπλούς σταθμούς ή από ένα σταθμό διαφορετικό από τον παρόντα, τότε η αντίστοιχη σελίδα ακυρώνεται εάν φυλάσσονταν στη λανθάνουσα μνήμη. Κατόπιν ο σταθμός ελέγχει τις διευθύνσεις των (ισχυρά) τροποποιημένων σελίδων, μαζί με τους αντίστοιχους τροποποιητές, οι οποίοι και θα γίνουν οι αντίστοιχες νέες έδρες, ως εξής:

   (α′) Εάν ο σταθμός είναι η νέα έδρα της μετακινηθείσας σελίδας, τότε: Εάν ο σταθμός δεν ήταν ο μοναδικός τροποποιητής της και άρα δεν έχει έγκυρο αντίγραφο της σελίδας, ζητάει τη σελίδα από την έως τότε έδρα της. Εάν ο σταθμός ήταν ο μοναδικός τροποποιητής της και έχει ήδη ένα έγκυρο αντίγραφό της στη λανθάνουσα μνήμη του, δε χρειάζεται να τη ζητήσει. Δίνεται μια θέση για τη σελίδα αυτή στον πίνακα των σελίδων που εδράζονται στο σταθμό αυτό, η θέση της στον πίνακα των σελίδων που κρατούνται στη λανθάνουσα μνήμη αποδεσμεύεται και το πεδίο που αναφέρει το σταθμό-έδρα της σελίδας αυτής στην αντίστοιχη θέση του καθολικού πίνακα σελίδων τίθεται να δείχνει στη νέα έδρα.

   (β′) Εάν ο σταθμός είναι η παλαιά έδρα της μετακινηθείσας σελίδας, τότε η αντίστοιχη θέση της στον πίνακα των σελίδων που εδράζονται στο σταθμό αυτό αποδεσμεύεται, η σελίδα ακυρώνεται εάν δεν υπάρχει ελεύθερη θέση στον πίνακα των σελίδων που κρατούνται στη λανθάνουσα μνήμη ή κρατείται ως έγκυρη εάν βρεθεί ελεύθερη θέση

για τη σελίδα αυτή στον πίνακα αυτό και το πεδίο που αναφέρει το σταθμό-έδρα της σελίδας αυτής στην αντίστοιχη θέση του καθολικού πίνακα σελίδων τίθεται να δείχνει στη νέα έδρα.

(γ′) Εάν ο σταθμός δεν είναι ούτε η νέα ούτε η παλαιά έδρα της μετακινηθείσας σελίδας, τότε μόνο το πεδίο που αναφέρει το σταθμό-έδρα της σελίδας αυτής στην αντίστοιχη θέση του καθολικού πίνακα σελίδων τίθεται να δείχνει στη νέα έδρα.

Ο γενικευμένος αυτός αλγόριθμος μετανάστευσης σελίδων βασίζεται — κατά το δυνατόν— στον ήδη υπάρχοντα και προσπαθεί πάλι να απαιτεί τα ελάχιστα δυνατά επιπλέον μηνύματα για τη μετανάστευση. Έτσι οι πληροφορίες που αφορούν τη μετανάστευση φορτώνονται στα ήδη υπάρχοντα μηνύματα αίτησης και παραχώρησης του φράγματος. Στην περίπτωση του μοναδικού εγγραφέα μιας σελίδας, ο οποίος έχει ήδη έγκυρο αντίγραφο της στη λανθάνουσα μνήμη του, η σελίδα μεταναστεύει σε αυτόν χωρίς να μεταδοθεί ξανά από την έδρα της. Στην περίπτωση που η σελίδα έχει τροποποιηθεί από πολλούς σταθμούς πρέπει —αναπόφευκτα— ο ισχυρότερος τροποποιητής να τη λάβει από την προηγούμενη έδρα της, στην οποία —και μόνο— έχουν εφαρμοσθεί όλες οι διαφορές.

3.4 Υλοποίηση του νέου μηχανισμού μετανάστευσης σελίδων

3.4.1 Πρακτικές ρυθμίσεις και μετατροπές για την ασφαλή λειτουργία του JIAJIA

Το JIAJIA τρέχει σε αυτόνομους σταθμούς εργασίας ή σε σταθμούς εργασίας που χρησιμοποιούν Network File System — NFS. Σε ένα αρχείο ρυθμίσεων

με όνομα .jiahosts ορίζεται ένας κόμβος ως διαχειριστής (master) και οι υπόλοιποι ως εργάτες (slaves). Η εκάστοτε εφαρμογή που περιλαμβάνει στον πηγαίο κώδικά της κλήσεις προς τη βιβλιοθήκη του JIAJIA εκτελείται από το χρήστη στον κόμβο - διαχειριστή και αυτόματα από το JIAJIA στους κόμβους - εργάτες. Όλες οι εργασίες εισόδου/εξόδου λαμβάνουν χώρα κανονικά στο διαχειριστή, ενώ στους εργάτες οι έξοδοι (κανονικές και σφαλμάτων) ανακατευθύνονται σε αρχεία.

Στην περίπτωση που οι κόμβοι δε χρησιμοποιούν NFS η επικοινωνία γίνεται στο πρωτότυπο JIAJIA με το rcp για την απομακρυσμένη αντιγραφή αρχείων και με το rexec για την απομακρυσμένη εκτέλεση διεργασιών. Για λόγους ασφαλείας τα παραπάνω αντικαταστάθηκαν με τα scp και ssh αντίστοιχα, επεμβαίνοντας στον πηγαίο κώδικα. Τα τελευταία προσφέρουν κρυπτογραφημένη μετάδοση δεδομένων. Κατά τις επεμβάσεις λήφθησαν υπ΄ όψιν οι λεπτομέρειες της διεπαφής των scp και ssh με το χρήστη. Επίσης κάποιες εσωτερικές χρονικές αναμονές του JIAJIA αναπροσαρμόστηκαν, ώστε να λαμβάνεται υπ΄ όψιν ο επιπλέον χρόνος που απαιτείται για το πρωτόκολλο χειραψίας του SSH και για τη μετάδοση των κρυπτογραφημένων πλέον πληροφοριών. Το SSH ρυθμίστηκε έτσι, ώστε να μην απαιτείται από το χρήστη η επίπονη διαδικασία της εισαγωγής κωδικού για κάθε απομακρυσμένη λειτουργία, αντιγράφοντας τις ίδιες ταυτότητες και κλειδιά σε όλους τους σταθμούς ή χρησιμοποιώντας πράκτορα πιστοποίησης.

Το JIAJIA περιέχει βασικά τον πηγαίο κώδικα της βιβλιοθήκης, τον πηγαίο κώδικα ορισμένων πρότυπων εφαρμογών και βιβλιογραφία. Περιλαμβάνονται και αντίστοιχα Makefiles για τις υποστηριζόμενες πλατφόρμες για να μεταγλωττισθούν η βιβλιοθήκη (libjia.a) και οι εφαρμογές που την καλούν.

Οι συναρτήσεις της βιβλιοθήκης JIAJIA μπορούν να χρησιμοποιηθούν από εφαρμογές γραμμένες σε C, αλλά και σε FORTRAN. Πιο συγκεκριμένα, μ-

πορούν να κληθούν από προγράμματα Fortran 77 εφόσον χρησιμοποιηθούν μεταγλωττιστές που υποστηρίζουν τη δήλωση POINTER.

3.4.2 Η διαχείριση της κοινής μνήμης στο JIAJIA

Οι πίνακες σελίδων της κοινής μνήμης στο JIAJIA

Η βασισμένη σε έδρες αρχιτεκτονική του JIAJIA καθιστά τη διαχείριση της κοινής μνήμης σχετικά απλή. Χρησιμοποιούνται τρεις βασικοί πίνακες σελίδων (δομών) σε κάθε σταθμό, εκ των οποίων μόνο ένας (ο καθολικός πίνακας σελίδων) είναι ίδιος για όλους τους σταθμούς.

Ο καθολικός πίνακας σελίδων απαιτεί μόνο 6 bytes για κάθε σελίδα (δομή `jiapage_t`), τα οποία περιλαμβάνουν έναν αναγνωριστή του σταθμού-έδρας της σελίδας, έναν αριθμοδείκτη (index) για τη θέση της σελίδας αυτής στον πίνακα των εδραζόμενων σελίδων και έναν αριθμοδείκτη για τη θέση της σελίδας αυτής στον πίνακα των σελίδων που κρατούνται στη λανθάνουσα μνήμη. Η δομή μιας σελίδας του καθολικού πίνακα σελίδων έχει ως εξής:

```
typedef struct{
           unsigned short int cachei;
           unsigned short int homei;
           unsigned short int homepid;
              } jiapage_t;
```

Ο πίνακας των σελίδων που εδράζονται σε κάθε σταθμό περιλαμβάνει — μεταξύ άλλων— για κάθε σελίδα (δομή `jiahome_t`) την καθολική της διεύθυνση,

μία ετικέτα εγγραφής, που δηλώνει εάν η σελίδα έχει τροποποιηθεί από τον ίδιο το σταθμό-έδρα και μία ετικέτα ανάγνωσης, που δηλώνει εάν η σελίδα έχει αποθηκευθεί στη λανθάνουσα μνήμη άλλων σταθμών. Η δομή μιας σελίδας που εδράζεται σε ένα σταθμό έχει ως εξής:

```
typedef struct{
          char              wtnt;
          char              rdnt;
          address_t         addr;
          wtvect_t          *wtvect;  /* used only for write vector */
          address_t         twin;     /* used only for write vector */
          char              wvlazy;   /* used only for write vector */
             } jiahome_t;
```

Ο πίνακας των σελίδων που κρατούνται στη λανθάνουσα μνήμη κάθε σταθμού περιλαμβάνει —μεταξύ άλλων— για κάθε σελίδα (δομή `jiacache_t`) τη διεύθυνσή της, την κατάσταση προστασίας της και ένα δείκτη στη δίδυμή της. Η δομή μιας σελίδας αποθηκευόμενης στη λανθάνουσα μνήμη ενός σταθμού έχει ως εξής:

```
typedef struct{
          pagestate_t       state;
          address_t         addr;
          address_t         twin;
          char              wtnt;
             } jiacache_t;
```

Στο JIAJIA κάθε κοινή σελίδα μένει πάντα στην ίδια διεύθυνση χρήστη σε όλους τους σταθμούς και ανεξάρτητα από το αν πρόκειται για σελίδα που κρατείται στην έδρα της ή για σελίδα που φυλάσσεται προσωρινά στη λανθάνουσα μνήμη κάποιου σταθμού. Το αποτέλεσμα του γεγονότος αυτού είναι απλότητα στην υλοποίηση, μιας και δεν απαιτείται μετάφραση διευθύνσεων μεταξύ των διαφόρων σταθμών.

Η κατανομή της κοινής μνήμης στο JIAJIA

Η κατανομή (allocation) κοινής μνήμης στο JIAJIA επιτρέπει το σαφή έλεγχο της κατανομής των κοινών δεδομένων και γίνεται μέσω της συνάρτησης `unsigned long jia_alloc3(int size, int block, int starthost)` της διεπαφής του με τον προγραμματιστή εφαρμογών, ή με κάποια απλοποιημένη μορφή της [12]. Η συναρτήσεις αυτές χρησιμοποιούν την κλήση συστήματος `mmap()` του UNIX. Η αντιστοίχιση (mapping) γίνεται σε συγκεκριμένες διευθύνσεις με τις παραμέτρους `MAP_FIXED|MAP_PRIVATE` της `mmap()`. Μία κοινή σελίδα αντιστοιχίζεται αρχικά μόνο στο σταθμό που αποτελεί την έδρα της. Αρχικά αντιστοιχίζεται με κατάσταση προστασίας (protection mode) `PROT_READ` μόνο, ώστε να ανιχνευθούν εγγραφές από την έδρα της. Εάν έχουμε μόνο ένα σταθμό υπάρχει βελτιστοποίηση, ώστε οι κοινές σελίδες να αντιστοιχίζονται ως `PROT_READ|PROT_WRITE` εξ αρχής.

Ο χειρισμός των SIGSEGVs στο JIAJIA

Προσβάσεις σε περιοχές της μνήμης που δεν έχουν την κατάλληλη κατάσταση προστασίας έχουν ως αποτέλεσμα τη δημιουργία ενός σήματος τύπου `SIGSEGV` (Segmentation Violation Signal — σήμα παραβίασης κατάτμησης).

Τη σύλληψη και εξυπηρέτηση των σημάτων αυτών αναλαμβάνει ο χειριστής τους (SIGSEGV handler). Αρχικά βρίσκει τη διεύθυνση μνήμης στην οποία προκλήθηκε το σφάλμα και την κατάσταση σφάλματος. Εάν η διεύθυνση υποδεικνύει ότι το σφάλμα πρόσβασης συνέβη στην έδρα της σελίδας, τότε πρέπει να έχουμε την περίπτωση ενός σφάλματος εγγραφής, μιας και όλες οι εντός έδρας σελίδες αντιστοιχούνται ως PROT_READ ή τίθενται ως PROT_READ στην αρχή ενός διαστήματος. Σε αυτήν την περίπτωση ο χειριστής απλώς αλλάζει την κατάσταση προστασίας της σελίδας σε PROT_READ|PROT_WRITE και καταγράφει το γεγονός ότι η σελίδα έχει τροποποιηθεί.

Εάν ο σταθμός δεν είναι η έδρα της σελίδας, δηλαδή ο σταθμός πρέπει να προσπελάσει μια κοινή σελίδα η οποία δεν είναι αντιστοιχημένη σε αυτόν, τη φέρνει από την έδρα της και την αντιστοιχεί τοπικά (στην ίδια διεύθυνση που είναι αντιστοιχημένη και στην έδρα της). Προκειμένου να αποθηκευθεί η σελίδα βρίσκεται μια θέση στη λανθάνουσα μνήμη ως εξής: Αρχικά αναζητείται μία σελίδα αποθηκευμένη στη λανθάνουσα μνήμη με την ίδια διεύθυνση με αυτή που προκάλεσε το σφάλμα (για την περίπτωση ενός σφάλματος ανάγνωσης σε μία άκυρη σελίδα ή ενός σφάλματος εγγραφής σε μία άκυρη ή μόνο-ανάγνωσης σελίδα). Στη συνέχεια —και εφόσον δεν υπάρχει σελίδα με τη διεύθυνση αυτή— αναζητείται μια αποαντιστοιχημένη σελίδα. Ακολούθως —και εφόσον δεν υπάρχει ούτε αποαντιστοιχημένη σελίδα— αναζητείται μια άκυρη σελίδα προκειμένου να αντικατασταθεί. Εάν όλες οι παραπάνω μέθοδοι αποτύχουν να βρουν μια θέση στη λανθάνουσα μνήμη για τη σελίδα, μία σελίδα μόνο - ανάγνωσης ή ανάγνωσης - εγγραφής αντικαθίσταται τυχαία ή με κυκλικό (round-robin) τρόπο, προκειμένου να βρεθεί χώρος. Όταν αντικαθίσταται μια εγγράψιμη σελίδα, η σχετική διαφορά και ειδοποίηση εγγραφής καταγράφονται για να παραδοθούν στην έδρα της σελίδας στην απελευθέρωση του σχετικού κλειδιού ή φράγματος.

Αφού βρεθεί μια θέση στη λανθάνουσα μνήμη για τη σελίδα που προκάλεσε το σφάλμα συμβαίνουν τα εξής: Η κατάσταση της σελίδας αυτής τίθεται σε

PROT_READ|PROT_WRITE εάν η σελίδα έχει ήδη αντιστοιχηθεί τοπικά (σε περίπτωση ενός σφάλματος ανάγνωσης σε μία άκυρη σελίδα ή ενός σφάλματος εγγραφής σε μία άκυρη ή μόνο-ανάγνωσης σελίδα). Διαφορετικά η σελίδα αντιστοιχείται στη διεύθυνση που προκάλεσε το σφάλμα ως PROT_READ|PROT_WRITE.

Στην περίπτωση ενός σφάλματος εγγραφής μία αίτηση λήψης σελίδας (GETP) στέλνεται στην έδρα της σελίδας (εκτός και εάν πρόκειται για σφάλμα εγγραφής σε σελίδα μόνο-ανάγνωσης, το οποίο εξυπηρετείται τοπικά). Όταν παραληφθεί η παραχώρηση λήψης σελίδας (GETPGRANT), η σελίδα που προκάλεσε το σφάλμα αντιγράφεται στην αντίστοιχη διεύθυνση. Η λανθάνουσα μνήμη καταγράφει στη συνέχεια τη διεύθυνση και την κατάσταση της σελίδας. Ένα *δίδυμο (twin)* — πανομοιότυπο αντίγραφο της σελίδας — δημιουργείται και φυλάσσεται στη λανθάνουσα μνήμη, ώστε να μπορούν να εντοπισθούν οι αλλαγές που θα υποστεί η σελίδα και να κωδικοποιηθούν σε διαφορές (diffs).

Στην περίπτωση ενός σφάλματος ανάγνωσης ακολουθείται η ίδια διαδικασία, με τη διαφορά ότι η κατάσταση προστασίας της νεοφερμένης σελίδας τίθεται να είναι PROT_READ και δε χρειάζεται να δημιουργηθεί δίδυμο της σελίδας.

Εξυπηρέτες σχετιζόμενοι με τη διαχείριση της κοινής μνήμης στο JIAJIA

Το τμήμα του JIAJIA που ασχολείται με την επικοινωνία των σταθμών προσφέρει αξιόπιστη και σε σωστή σειρά παράδοση μηνυμάτων, πάνω από το πρωτόκολλο UDP/IP. Η συνάρτηση ασύγχρονης αποστολής μηνυμάτων `void asendmsg(jia_msg_t *msg)` στέλνει ένα μήνυμα στο σταθμό που ορίζεται στο αντίστοιχο πεδίο του μηνύματος. Φθάνοντας το μήνυμα στο σταθμό προορισμού του προκαλεί εκεί τη δημιουργία ενός σήματος τύπου SIGIO (In/Out Signal — σήμα εισόδου/εξόδου). Τη σύλληψη και εξυπηρέτηση των σημάτων αυτών αναλαμβάνει ο χειριστής τους

(SIGIO handler), ο οποίος αναλαμβάνει να καλέσει έναν εξυπηρέτη μηνυμάτων (`void msgserver()`), ο οποίος με τη σειρά του ανάλογα με τον τύπο του μηνύματος καλεί τον αντίστοιχο εξυπηρέτη.

Οι βασικοί τύποι μηνυμάτων που σχετίζονται με τη διαχείριση της κοινής μνήμης στο JIAJIA είναι οι εξής: Τα μηνύματα αίτησης (`GETP`) και παραχώρησης (`GETPGRANT`) λήψης σελίδας, τα μηνύματα διαφορών (`DIFF`) και παραχώρησης διαφορών (`DIFFGRANT`) και τα μηνύματα ειδοποίησης εγγραφής (`WTNT`) και ακύρωσης (`INVLD`). Τα δύο τελευταία χρησιμοποιούνται μόνο όταν τα μηνύματα ειδοποίησης εγγραφής είναι περισσότερα από όσα μπορούν να χωρέσουν σε ένα μήνυμα αντίστοιχα αίτησης ή παραχώρησης κλειδιού ή φράγματος.

Λαμβάνοντας ένα μήνυμα αίτησης λήψης σελίδας (`GETP`) ο αντίστοιχος εξυπηρέτης (`void getpserver(jia_msg_t *req)`), που βρίσκεται στην έδρα της σελίδας που ζητείται, ενεργεί ως εξής: Παίρνει τη διεύθυνση της ζητούμενης σελίδας από το μήνυμα της αίτησης, διαβάζει τη σελίδα από τη μνήμη και τη στέλνει στο σταθμό που τη ζήτησε, στο μήνυμα παραχώρησης λήψης σελίδας (`GETPGRANT`). Ο σταθμός εκείνος την παραλαμβάνει με τη σειρά του μέσω του αντίστοιχου εξυπηρέτη (`void getpgrantserver(jia_msg_t *rep)`).

Λαμβάνοντας ένα μήνυμα διαφορών (`DIFF`) ο αντίστοιχος εξυπηρέτης (`void diffserver(jia_msg_t *req)`), που βρίσκεται στην έδρα των σελίδων στις οποίες αναφέρονται οι διαφορές, ενεργεί ως εξής: Παίρνει διαδοχικά κάθε διεύθυνση σελίδας και διαφορά που περιέχεται στο μήνυμα, αποκωδικοποιεί τη διαφορά και κάνει την εφαρμογή στην αντίστοιχη σελίδα μνήμης. Συγκεκριμένα, η σελίδα στην οποία αναφέρεται η εκάστοτε διαφορά γίνεται εγγράψιμη πριν εφαρμοσθεί η διαφορά και κατόπιν επανατίθεται στην προηγούμενη κατάστασή της. Στη συνέχεια στέλνει ένα μήνυμα παραχώρησης διαφορών (`DIFFGRANT`), για επιβεβαίωση, στο σταθμό που έστειλε τις διαφορές. Αυτός το λαμβάνει μέσω του αντίστοιχου εξυπηρέτη (`void diffgrantserver(jia_msg_t *rep)`).

3.4.3 Το βασισμένο σε κλειδιά πρωτόκολλο συνοχής της λανθάνουσας μνήμης του JIAJIA

Προκειμένου να διατηρεί συνεπή τη λανθάνουσα μνήμη το JIAJIA υλοποιεί τη συνέπεια εμβέλειας (scope consistency — ScC), η οποία είναι πιο χαλαρή από τη συνέπεια απελευθέρωσης (release consistency — RC). Η συνέπεια εμβέλειας είναι απλούστερη της συνέπειας αδρανούς απελευθέρωσης (Lazy Release Consistency — LRC), στο ότι δεν απαιτεί την υλοποίηση της πλήρους happened-before-1 σειράς [18]. Συγκεκριμένα απαιτεί μόνο τα δεδομένα προηγούμενων διαστημάτων που σχετίζονται με το αποκτηθέν κλειδί να είναι ορατά στον επεξεργαστή που αποκτά το κλειδί. Το ιδιαίτερο χαρακτηριστικό του πρωτοκόλλου συνοχής της λανθάνουσας μνήμης είναι ότι είναι *βασισμένο σε κλειδιά αμοιβαίου αποκλεισμού (lock - based)* . Δηλαδή δε βασίζεται σε πληροφορίες καταλόγου για τη διατήρηση της συνοχής, αλλά αυτή επιτυγχάνεται μέσω ειδοποιήσεων εγγραφής που κρατούνται στο κλειδί. Συνοπτικά το πρωτόκολλο έχει ως εξής [14]:

Κάθε σελίδα έχει μία καθορισμένη έδρα και μπορεί να φυλαχθεί σε κάποια λανθάνουσα μνήμη εκτός έδρας σε μία από τρεις καταστάσεις: Άκυρη, Μόνο - Ανάγνωσης και Ανάγνωσης - Εγγραφής. Ως ένα ειδικό κοινό αντικείμενο, κάθε κλειδί αμοιβαίου αποκλεισμού έχει επίσης μία έδρα.

Σε μία απελευθέρωση κλειδιού, ο επεξεργαστής που το απελευθερώνει συγκρίνει όλες τις σελίδες που είναι αποθηκευμένες στη λανθάνουσα μνήμη του και που γράφτηκαν στον τελευταίο κρίσιμο τομέα με τα δίδυμά τους, προκειμένου να εξάγει τις διαφορές του κρίσιμου αυτού τομέα. Οι διαφορές αυτές στέλνονται στη συνέχεια στις αντίστοιχες έδρες των σελίδων. Όταν όλες οι διαφορές έχουν εφαρμοσθεί στις σελίδες (στις έδρες τους), ένα μήνυμα απελευθέρωσης στέλνεται στο διαχειριστή του συγκεκριμένου κλειδιού, προκειμένου αυτός να

απελευθερώσει το κλειδί. Επιπλέον, ο επεξεργαστής που απελευθερώνει το κλειδί φορτώνει τις ειδοποιήσεις εγγραφών του συγκεκριμένου κρίσιμου τομέα στο μήνυμα απελευθέρωσης, ειδοποιώντας για τις σελίδες που τροποποιήθηκαν στο συγκεκριμένο κρίσιμο τομέα.

Σε μία απόκτηση κλειδιού, ο επεξεργαστής που πρόκειται να το αποκτήσει στέλνει μία αίτηση απόκτησης κλειδιού στο διαχειριστή του συγκεκριμένου κλειδιού. Ο επεξεργαστής που ζητά το κλειδί αδρανοποιείται στη συνέχεια, μέχρι αυτό να του παραχωρηθεί. Ο διαχειριστής του κλειδιού φορτώνει στο μήνυμα παραχώρησης τις ειδοποιήσεις εγγραφών που σχετίζονται με το συγκεκριμένο κλειδί. Όταν ο επεξεργαστής που ζήτησε το κλειδί λάβει το μήνυμα παραχώρησης, ακυρώνει όλες τις σελίδες της λανθάνουσας μνήμης του για τις οποίες ενημερώθηκε από τις σχετικές ειδοποιήσεις εγγραφών ότι είναι πλέον απαρχαιωμένες.

Ένα φράγμα μπορεί να ειδωθεί ως ένας συνδυασμός μιας απελευθέρωσης και μιας απόκτησης κλειδιού. Η άφιξη σε ένα φράγμα σηματοδοτεί το τέλος ενός παλιού κρίσιμου τομέα, ενώ η αναχώρηση από το φράγμα σημειώνει την αρχή ενός νέου κρίσιμου τομέα. Κατ΄ αυτόν τον τρόπο, δύο φράγματα περιβάλλουν έναν κρίσιμο τομέα. Σε ένα φράγμα, όλες οι ειδοποιήσεις εγγραφών όλων των κλειδιών καθαρίζονται, δηλαδή στέλνονται σε όλους τους επεξεργαστές για να ακυρώσουν τις αντίστοιχες σελίδες.

Σε μία αστοχία ανάγνωσης, η σελίδα που προκάλεσε το σφάλμα μεταφέρεται στην τοπική μνήμη από την έδρα της, σε κατάσταση μόνο - ανάγνωσης.

Σε μία αστοχία εγγραφής, εάν η σελίδα που πρόκειται να γραφεί δεν είναι παρούσα ή είναι σε άκυρη κατάσταση στην τοπική μνήμη, μεταφέρεται από την έδρα της σε κατάσταση ανάγνωσης - εγγραφής. Εάν η σελίδα που πρόκειται να γραφεί είναι σε κατάσταση μόνο - ανάγνωσης στην τοπική μνήμη, η κατάσταση

γίνεται ανάγνωσης - εγγραφής. Μία ειδοποίηση εγγραφής καταγράφεται για τη σελίδα αυτή και ένα δίδυμό της δημιουργείται πριν αυτή γραφεί.

Κατά την αντικατάσταση μιας σελίδας που βρίσκεται στη λανθάνουσα μνήμη, η σελίδα αυτή γράφεται στην έδρα της, στην περίπτωση που ήταν αποθηκευμένη σε κατάσταση ανάγνωσης - εγγραφής στη λανθάνουσα μνήμη. Ομοίως και οι διαφορές σελίδων της λανθάνουσας μνήμης που βρίσκονται σε κατάσταση ανάγνωσης - εγγραφής και πρόκειται να ακυρωθούν γράφονται στις έδρες των αντίστοιχων σελίδων πριν οι σελίδες ακυρωθούν.

3.4.4 Μέτρηση και αποθήκευση των απομακρυσμένων διαφορών

Μιας και το JIAJIA είναι μία κατανεμημένη κοινή μνήμη λογισμικού βασισμένη σε έδρες, κάθε σελίδα εδράζεται σε ένα σταθμό. Όλες οι αλλαγές που κάνουν οι σταθμοί σε οποιαδήποτε σελίδα που φιλοξενούν εκτός έδρας μεταφέρονται ξανά στην έδρα της με τη μορφή διαφορών. Στη συνέχεια οι διαφορές αυτές εφαρμόζονται εντός έδρας, ώστε η έδρα να έχει τη σωστή έκδοση των σελίδων που της ανήκουν. Συνεπώς η έδρα είναι και η πλέον κατάλληλη για να μετράει όλες τις τροποποιήσεις από όλους τους σταθμούς των σελίδων που της ανήκουν, μετρώντας τα bytes των διαφορών που εφαρμόζονται σε κάθε σελίδα.

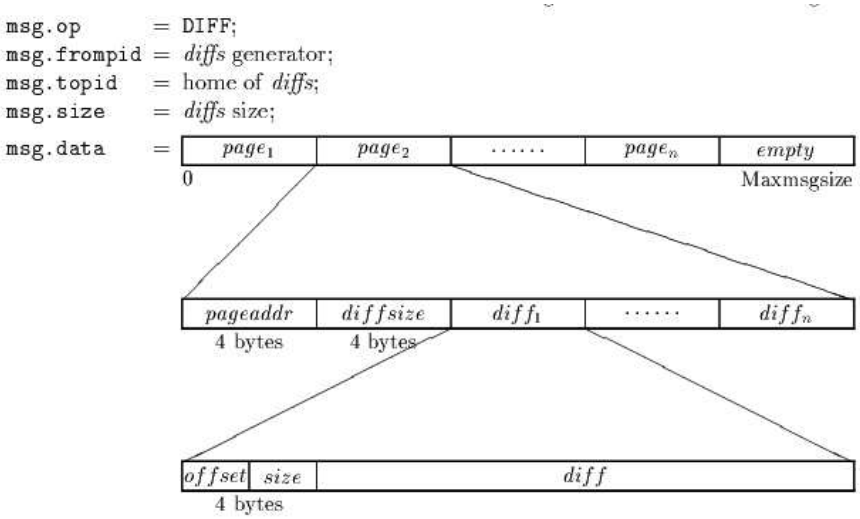
Προκειμένου να αποθηκεύεται η λογιστική αυτή πληροφορία προστέθηκε στη δομή κάθε σελίδας που εδράζεται σε ένα σταθμό έναν πίνακα (`unsigned long diffsizes[Maxhosts]`), με μέγεθος ίσο με το μέγιστο δυνατό αριθμό σταθμών. Δε σπαταλάται πολλή μνήμη, ώστε να υπήρχε ανάγκη δυναμικού ορισμού του πίνακα. Κάθε θέση του πίνακα αυτού αντιστοιχεί σε ένα σταθμό και σε κάθε τέτοια θέση αποθηκεύεται *αθροιστικά* ο αριθμός των bytes των διαφορών που εφαρμόζονται στη σελίδα αυτή από τον αντίστοιχο σταθμό.

Κατά την απελευθέρωση ενός κλειδιού (ή φθάνοντας σε ένα φράγμα), τελειώνοντας δηλαδή έναν κρίσιμο τομέα, ο σταθμός που απελευθερώνει το κλειδί (ή φθάνει στο φράγμα) συγκρίνει όλες τις σελίδες που έχει αποθηκευμένες στη λανθάνουσα μνήμη του και που γράφτηκαν στον κρίσιμο αυτό τομέα με τα δίδυμα αντίγραφα τους. Οι δίδυμες αυτές σελίδες είναι αντίγραφα των σελίδων όπως εκείνες ήταν στην αρχική τους μορφή, όταν είχαν σταλεί στο σταθμό από την έδρα τους. Από τη σύγκριση αυτή προκύπτουν οι διαφορές των σελίδων για τον κρίσιμο αυτό τομέα.

Οι διαφορές αυτές κατασκευάζονται στο JIAJIA από τη συνάρτηση `int encodediff(int cachei, unsigned char* diff)`, τοποθετούνται σε μηνύματα (με παραλήπτη την έδρα της εκάστοτε σελίδας) από τη συνάρτηση `void savediff(int cachei)` και τα μηνύματα αυτά στέλνονται από τη συνάρτηση `void senddiffs()`. Κάθε μήνυμα περιέχει όσο περισσότερες διαφορές σελίδων χωράει —και υπάρχουν, οι οποίες αφορούν σελίδες με την ίδια έδρα. Η δομή κάθε μηνύματος, το οποίο και απευθύνεται σε μία συγκεκριμένη έδρα κάθε φορά, φαίνεται στο σχήμα 3.1. Παρατηρούμε ότι για κάθε σελίδα μπορεί να υπάρχουν παραπάνω από μία διαφορές.

Από τη διαδικασία της κωδικοποίησης των διαφορών ο κατασκευαστής αυτών είναι σε θέση να γνωρίζει το μέγεθος τους και μάλιστα το ενσωματώνει και στα δεδομένα που στέλνει στην έδρα των αντίστοιχων σελίδων. Πρόκειται για τον `diffsize`, έναν αριθμό 4 bytes ο οποίος αναφέρεται στο συνολικό μέγεθος όλων των διαφορών για μια συγκεκριμένη σελίδα. Συνεπώς δεν απαιτείται να προστεθεί επιπλέον πεδίο στο μήνυμα τύπου `DIFF` για να αποθηκεύονται τα μεγέθη των διαφορών, παρά μπορεί ο παραλήπτης του μηνύματος, η έδρα δηλαδή των σελίδων, να διαβάζει την πληροφορία αυτή από τα δεδομένα του μηνύματος.

Ο `diffserver()` τροποποιήθηκε, ώστε να διαβάζει από τα δεδομένα κάθε μηνύματος διαφορών την πληροφορία του μεγέθους των διαφορών για κάθε σελίδα και να την προσθέτει στη θέση του πίνακα `diffsizes[]` που αντιστοιχεί στο



Σχήμα 3.1: Η δομή των μηνυμάτων τύπου DIFF [13].

σταθμό που ήταν αποστολέας του μηνύματος διαφορών. Το μέγεθος των διαφορών που διαβάζει ο `diffserver()` επιβεβαιώθηκε ότι είναι το ίδιο με εκείνο που είχε υπολογίσει και ο δημιουργός των διαφορών με την `encodediff()`. Επιπλέον επιβεβαιώθηκε ότι και σε άλλα σημεία του κώδικα, όπως για παράδειγμα φθάνοντας σε ένα φράγμα, τα μεγέθη των διαφορών παραμένουν σωστά.

Οι μόνες διαφορές το μέγεθος των οποίων δε μετράται με την παραπάνω διαδικασία είναι εκείνες που προκύπτουν από τοπικές αλλαγές σε σελίδες εντός έδρας. Για τις σελίδες αυτές δε δημιουργούνται δίδυμες και διαφορές, αφού κάτι τέτοιο θα ήταν άσκοπη σπατάλη υπολογιστικού χρόνου και θα επέφερε χαμηλές επιδόσεις. Μοναδική ένδειξη για προσβάσεις σε τοπικές σελίδες (οι οποίες εξυπηρετούνται απ΄ ευθείας από το λειτουργικό σύστημα χωρίς να μεσολαβεί η κατανεμημένη κοινή μνήμη λογισμικού) θα μπορούσε να δοθεί από την παρακολούθηση των ενσωματωμένων στον επεξεργαστή μετρητών επίδοσης. Οι τοπικές προσπελάσεις και τροποποιήσεις λήφθησαν υπ΄ όψιν μόνο έμμεσα, στον ορισμό των κατωφλιών μετανάστευσης για κάθε εφαρμογή. Τα κατώφλια αυτά βοηθούν στην αποφυγή της παλινδρόμησης σελίδων μεταξύ των ίδιων

σταθμών (ping-pong), που θα επέφερε μείωση της απόδοσης, λόγω περιττής επικοινωνιακής και υπολογιστικής επιβάρυνσης.

3.4.5 Επιλογή του σημείου λήψης των αποφάσεων μετανάστευσης και του τρόπου μετάδοσής τους

Εφόσον το JIAJIA χρησιμοποιεί συνέπεια εμβέλειας και το πρωτόκολλο συνοχής του είναι βασισμένο σε κλειδιά, οι πράξεις που έχουν να κάνουν με τη διατήρηση της συνοχής λαμβάνονται στα σημεία συγχρονισμού. Μιας και τα φράγματα είναι συνήθως σημαντικά σημεία συγχρονισμού των παράλληλων εφαρμογών, επιλέχθησαν αυτά ως σημεία λήψης όχι μόνο των αποφάσεων συνοχής, αλλά και των αποφάσεων μετανάστευσης σελίδων. Λαμβάνοντας και κοινοποιώντας τις αποφάσεις μετανάστευσης σελίδων στα φράγματα μειώνεται και το κόστος του μηχανισμού —ιδιαίτερα το επικοινωνιακό, αφού δίνεται η δυνατότητα της φόρτωσης των πληροφοριών που αφορούν τη μετανάστευση σελίδων στα ήδη υπάρχοντα μηνύματα αίτησης και απόκρισης φράγματος, όπως συμβαίνει και με τις πληροφορίες τις σχετικές με τη διατήρηση της συνοχής (τις ειδοποιήσεις εγγραφών).

Επιπλέον, ο εκάστοτε διαχειριστής του φράγματος αναλαμβάνει να συλλέξει και να μεταδώσει τις πληροφορίες μετανάστευσης σελίδων, όπως κάνει και με τις πληροφορίες διατήρησης της συνοχής (διαβάζοντάς τις φορτωμένες και φορτώνοντάς τις στα ήδη υπάρχοντα μηνύματα αίτησης και απόκρισης φράγματος αντίστοιχα). Αποφεύγεται έτσι η εκπομπή μηνυμάτων από όλους και προς όλους τους σταθμούς, η οποία θα ήταν πολύ ακριβή. Εφόσον ο διαχειριστής του φράγματος στο JIAJIA αλλάζει (με έναν απλό αλγόριθμο modulo) δεν υπάρχει και ο φόβος να εμφανισθεί κυκλοφοριακή συμφόρηση (bottleneck) σε ένα συγκεκριμένο σταθμό.

Με βάση τα παραπάνω παρατηρείται ότι τόσο οι ειδοποιήσεις εγγραφής, όσο και οι πληροφορίες μετανάστευσης σελίδων συλλέγονται από το διαχειριστή φράγματος και μεταδίδονται από αυτόν προς όλους τους σταθμούς με τον ίδιο τρόπο. Είναι σημαντικό όμως να μη συγχέονται οι ρόλοι των κατά τα άλλα ασύνδετων πληροφοριών: Οι ειδοποιήσεις εγγραφής αφορούν τις σελίδες που έχει κάθε σταθμός αποθηκευμένες στη λανθάνουσα μνήμη του, ενώ οι πληροφορίες μετανάστευσης σελίδων αφορούν τις σελίδες που εδράζονται στον κάθε σταθμό. Τέλος, ο μόνος λόγος που χρησιμοποιείται ο διαχειριστής φράγματος για τη συλλογή και μετάδοση όλων των πληροφοριών μετανάστευσης είναι η μείωση του επικοινωνιακού κόστους.

3.4.6 Υπολογισμός των μέγιστων τροποποιήσεων και του ισχυρότερου τροποποιητή

Εφόσον τα φράγματα ορίστηκε να είναι τα σημεία λήψης των αποφάσεων μετανάστευσης σελίδων, κάθε σταθμός φθάνοντας σε ένα φράγμα (αποτέλεσμα της κλήσης της συνάρτησης `void jia_barrier()` της διεπαφής του JIAJIA με τον προγραμματιστή εφαρμογών) και αφού στείλει τις ειδοποιήσεις εγγραφής πρέπει να υπολογίσει για κάθε εντός έδρας σελίδα το μέγεθος των μεγαλύτερων τροποποιήσεων και τον πιο ισχυρό τροποποιητή.

Κάθε σταθμός κρατά, όπως αναλύθηκε, για κάθε σελίδα που εδράζεται σε αυτόν τα bytes των διαφορών που εφαρμόζονται στη σελίδα από όλους τους σταθμούς. Μπορεί έτσι να υπολογίσει για κάθε εντός έδρας σελίδα το μέγεθος των μεγαλύτερων τροποποιήσεων και από ποιο σταθμό προήλθαν. Για την αποθήκευση της πληροφορίας αυτής προστέθηκαν στη δομή κάθε εντός έδρας σελίδας οι μεταβλητές `int hostwithmaxdiff` και `long maxdiffsize`. Υλοποιήθηκε μία νέα συνάρτηση, η `void migfindmax()`, την οποία εκτελεί

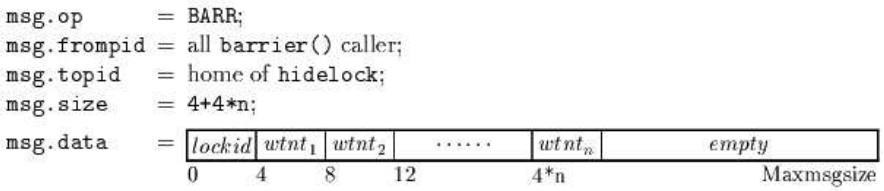
κάθε κόμβος προκειμένου να υπολογίζει το μέγεθος των μέγιστων τροποποιήσεων και τον πιο ισχυρό τροποποιητή για κάθε σελίδα που εδράζεται σε αυτόν και να τα αποθηκεύει στις αντίστοιχες μεταβλητές. Η συνάρτηση είναι απλή στην υλοποίησή της:

```
void migfindmax()
{
  /* For each homed page we get its hostwithmaxdiff and its maxdiffsize.
     We have no obvious way to measure local diffs, since no actual diffs
     are created from the local OS.  --repantis */

 int homei, hosti;

 for( homei=0 ; homei<Homepages ; homei++ )
  for( hosti=0 ; hosti<Maxhosts ; hosti++ ){
   if( home[homei].diffsizes[hosti] > home[homei].maxdiffsize ){
     home[homei].maxdiffsize = home[homei].diffsizes[hosti];
     home[homei].hostwithmaxdiff = hosti;
   }
  }
}
```

Οι μεταβλητές `maxdiffsize` και `hostwithmaxdiff` αρχικοποιούνται αντίστοιχα σε `0` και `-1` (έναν αδύνατο αναγνωριστή σταθμού). Η αρχικοποίηση γίνεται για όλες τις εντός έδρας σελίδες στη συνάρτηση αρχικοποίησης όλων των δομών που σχετίζονται με τη διαχείριση της μνήμης στο JIAJIA , την `void initmem()`. Εκεί έχει γίνει και η αρχικοποίηση σε 0 όλων των θέσεων των πίνακων `diffsizes[]` για όλες τις εντός έδρας σελίδες.

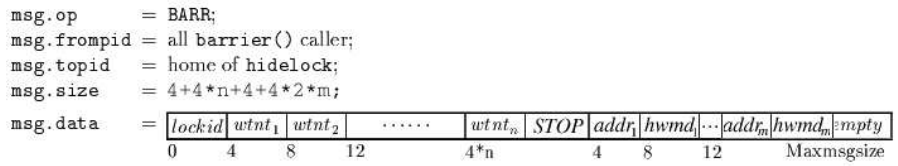


Σχήμα 3.2: Η αρχική δομή των μηνυμάτων τύπου BARR [13].

3.4.7 Αποστολή των πληροφοριών μετανάστευσης σελίδων μνήμης στο διαχειριστή φράγματος

Εφόσον ο εκάστοτε διαχειριστής φράγματος ορίστηκε να συλλέγει και να μεταδίδει τις πληροφορίες μετανάστευσης σελίδων, κάθε σταθμός μόλις τις συλλέξει πρέπει να τις μεταδώσει σε αυτόν. Με σκοπό τη μείωση του επικοινωνιακού κόστους οι πληροφορίες αυτές, όπως εξηγήθηκε, φορτώνονται στο σώμα του ήδη υπάρχοντος μήνυματος αίτησης φράγματος. Το μήνυμα αίτησης φράγματος του JIAJIA έχει κανονικά τη δομή που δείχνει το σχήμα 3.2. Παρατηρούμε ότι εκτός από τον αναγνωριστή του φράγματος που ζητείται περιλαμβάνονται και όσες ειδοποιήσεις εγγραφής χωράνε —και υπάρχουν.

Το σώμα του μηνύματος αίτησης φράγματος τροποποιήθηκε κατάλληλα, ώστε να φορτώνονται και οι πληροφορίες μετανάστευσης σελίδων. Συγκεκριμένα, προκειμένου να γίνεται σωστά η ανάλυση (parsing) του σώματος στον παραλήπτη, κρίθηκε αναγκαίο να διαχωρίζονται οι πληροφορίες μετανάστευσης από τις προηγούμενες πληροφορίες που περιέχονται στο σώμα του μηνύματος (όπως για παράδειγμα τις ειδοποιήσεις εγγραφής). Το ρόλο του διαχωριστή έχει ένα αλφαριθμητικό (πίνακας χαρακτήρων) μεγέθους 4 bytes (STOP), το οποίο υποδηλώνει το τέλος των προηγούμενων πληροφοριών και την αρχή των πληροφοριών μετανάστευσης σελίδων. Εφόσον δεν υπάρχει διεύθυνση μνήμης ίση με τον ASCII κωδικό του STOP δεν υπάρχει περίπτωση παρανόησης του διαχωριστή για ειδοποίηση εγγραφής από τον παραλήπτη του μηνύματος.



Σχήμα 3.3: Η τροποποιημένη δομή των μηνυμάτων τύπου BARR.

Προκειμένου να ολοκληρώνεται σωστά η ανάλυση του σώματος του μηνύματος από τον παραλήπτη, ο διαχωριστής προστίθεται ακόμη και στην περίπτωση που δεν έχουμε πληροφορίες μετανάστευσης σελίδων να φορτώσουμε στο μήνυμα. Σημειώνεται ότι η τροποποιημένη μορφή του μηνύματος αίτησης φράγματος χρησιμοποιείται μόνο όταν είναι ενεργοποιημένη η μετανάστευση σελίδων μνήμης.

Μία διαφορετική —αλλά πιθανά πιο πολύπλοκη— προσέγγιση θα ήταν ο παραλήπτης να γνωρίζει το μήκος των πληροφοριών που προηγούνται των πληροφοριών μετανάστευσης, μέσω ενός μεταδιδόμενου στο σώμα του μηνύματος αριθμού, ο οποίος θα έχει υπολογισθεί από τον αποστολέα.

Αμέσως μετά τους τέσσερις χαρακτήρες του διαχωριστή ακολουθούν οι πληροφορίες μετανάστευσης. Συγκεκριμένα, στην παρούσα έκδοση και για μια απλή πολιτική μετανάστευσης, μεταδίδονται διευθύνσεις μνήμης εντός έδρας σελίδων (4 bytes) και οι αντίστοιχοι αναγνωριστές των ισχυρότερων τροποποιητών των σελίδων αυτών (επίσης 4 bytes). Έτσι ο κάθε σταθμός ενημερώνει το διαχειριστή φράγματος για το ποιες σελίδες του τροποποιήθηκαν ισχυρά και από ποιο σταθμό, ώστε οι σελίδες αυτές να μεταναστεύσουν στους αντίστοιχους σταθμούς. Η δομή του τροποποιημένου μηνύματος αίτησης φράγματος δείχνεται στο σχήμα 3.3

Η προσάρτηση των πληροφοριών μετανάστευσης στο σώμα του μηνύματος αίτησης φράγματος γίνεται μέσω της συνάρτησης `void appendmsg(jia_msg_t`

*msg, unsigned char *str, int len) του JIAJIA . Στο σώμα της συνάρτησης αυτής γίνεται ο έλεγχος μήπως η προσάρτηση δημιουργεί μήνυμα μεγαλύτερο από το επιτρεπτό για το JIAJIA όριο. Εφόσον στις δοκιμές με συνήθεις εφαρμογές το όριο αυτό δεν ξεπεράστηκε, δεν παρουσιάσθηκε η ανάγκη για δημιουργία και επιπλέον τύπου μηνύματος, ειδικά για τη μετάδοση της πληροφορίας μετανάστευσης. (Αντίθετα υπάρχουν αποκλειστικά μηνύματα μετάδοσης ειδοποιήσεων εγγραφής προς (τύπου WTNT) και από (τύπου INVLD) το διαχειριστή κλειδιού ή φράγματος, όταν οι ειδοποιήσεις εγγραφής είναι πάρα πολλές για να χωρέσουν όλες στα μηνύματα αίτησης και παραχώρησης του κλειδιού ή του φράγματος.)

Προκειμένου να αποφευχθεί το κόστος των μεταναστεύσεων για σελίδες που τροποποιήθηκαν ελάχιστα από απομακρυσμένους σταθμούς, αλλά και για την αποφυγή του φαινομένου της παλινδρόμησης σελίδων μνήμης μεταξύ των ίδιων σταθμών ως έδρες, τίθεται κάποιο κατώφλι μεγέθους τροποποιήσεων. Έτσι δε φορτώνονται στο μήνυμα αίτησης φράγματος οι διευθύνσεις και οι μέγιστοι τροποποιητές όλων των τροποποιημένων σελίδων, αλλά μόνο εκείνων των οποίων το μέγεθος των τροποποιήσεων ξεπερνά το κατώφλι. Το κατώφλι αυτό μας επιτρέπει να λάβουμε έμμεσα υπ΄ όψιν και τις πιθανές τοπικές τροποποιήσεις από την έως τότε έδρα κάθε σελίδας, οι οποίες δεν ανιχνεύονται εύκολα και χωρίς πτώση της απόδοσης —πόσο μάλλον μετρούνται— (και είναι αόρατες για το μηχανισμό μέτρησης τροποποιήσεων που υλοποιήθηκε). Το κατώφλι μπορεί να αλλάζει ανάλογα με την εφαρμογή, το διασυνδετικό δίκτυο, τη μνήμη και την επεξεργαστική ισχύ των σταθμών και η τιμή του υπολογίζεται είτε εμπειρικά, είτε υλοποιώντας κάποιο μικρομετροπρόγραμμα (microbenchmark).

Προκειμένου ο προγραμματιστής εφαρμογών να μπορεί ο ίδιος να θέτει το κατώφλι πάνω από το οποίο θα διενεργείται μια μετανάστευση, προστέθηκε μια ακόμη επιλογή στη συνάρτηση jia_config() του API. Έτσι έχουμε την επιλογή jia_config(HMIGthreshold, int value), με την οποία μπορεί να

δωθεί τιμή στο κατώφλι. Η προεπιλεγμένη τιμή είναι «0», οπότε και προκαλείται μετανάστευση για οποιαδήποτε τιμή bytes απομακρυσμένης μεταβολής.

3.4.8 Αποθήκευση και επεξεργασία των πληροφοριών μετανάστευσης σελίδων μνήμης από το διαχειριστή φράγματος

Εφόσον ο εκάστοτε διαχειριστής φράγματος ορίστηκε να είναι αυτός που θα φροντίζει για τη συλλογή και για τη μετάδοση των πληροφοριών μετανάστευσης σελίδων, πρέπει να ενσωματώνει την κατάλληλη υποδομή για την αποθήκευση και την επεξεργασία των πληροφοριών αυτών. Προκειμένου ο διαχειριστής φράγματος να συλλέγει τις πληροφορίες μετανάστευσης σελίδων από τα μηνύματα αίτησης φράγματος, τροποποιήθηκε ο αντίστοιχος εξυπηρέτης, δηλαδή ο `void barrserver(jia_msg_t *req)`. Ο `barrserver()` καλείται για κάθε αίτηση φράγματος που λαμβάνει ο διαχειριστής φράγματος. Συνεπώς καλείται για ένα φράγμα τόσες φορές όσος είναι και ο αριθμός των σταθμών, μιας και το JIAJIA προσφέρει μόνο καθολικό φράγμα και όχι φράγμα για μερικούς σταθμούς μόνο. Προκειμένου να μεταδοθεί η απόκριση φράγματος καλείται — μόνο μία φορά, όταν έχουν φθάσει στο φράγμα όλοι οι σταθμοί— η αντίστοιχη συνάρτηση (`void grantbarr(long lock)`).

Εφόσον τα μηνύματα με πληροφορία μετανάστευσης που λαμβάνει ο διαχειριστής φράγματος είναι πολλά (όσα και ο αριθμός των σταθμών) δε θα ήταν σκόπιμο να πρέπει να διαβάσει ξανά η `grantbarr()` καθένα ξεχωριστά. Είναι άρα φανερό ότι ο `barrserver()` πρέπει να αποθηκεύει κάπου αθροιστικά τις πληροφορίες μετανάστευσης σελίδων που λαμβάνει από κάθε σταθμό, ώστε η `grantbarr()` να μπορεί κατόπιν να τις έχει συγκεντρωμένες για να τις φορτώσει στο μήνυμα παραχώρησης φράγματος. Επίσης, όπως είναι προφανές,

οι πληροφορίες μετανάστευσης σελίδων αναφέρονται σε συγκεκριμένο φράγμα κάθε φορά (δηλαδή κλειδί μιας και το φράγμα ισοδυναμεί με κλειδιά όπως εξηγήθηκε νωρίτερα).

Από τα παραπάνω συνάγεται ότι ο `barrserver()` πρέπει να αποθηκεύει συγκεντρωτικά όλες τις πληροφορίες μετανάστευσης σελίδων που λαμβάνει για κάποιο συγκεκριμένο φράγμα σε ένα επιπλέον πεδίο της δομής του κλειδιού που διαθέτει το JIAJIA και η οποία χρησιμοποιείται και για τα φράγματα. Οι πληροφορίες μετανάστευσης μεταφέρονται στο σώμα κάθε μηνύματος ως αλφαριθμητικά. Συνεπώς μπορούν να αποθηκευθούν σε ένα αλφαριθμητικό. Κάθε φράγμα έχει το δικό του αλφαριθμητικό για την αποθήκευση των πληροφοριών μετανάστευσης, το οποίο αποτελείται από τη συνένωση (concatenation) των επιμέρους πληροφοριών μετανάστευσης για το φράγμα αυτό από τους διάφορους κόμβους. Η απλή συνένωση των πληροφοριών δε δημιουργεί πρόβλημα, μιας και δεν υπάρχει περίπτωση να έλθει πληροφορία για την ίδια σελίδα από διαφορετικούς σταθμούς, αφού κάθε σελίδα εδράζεται σε ακριβώς ένα σταθμό.[1] Μιας και δεν απαιτείται επεξεργασία των πληροφοριών μετανάστευσης από το διαχειριστή φράγματος στο σημείο αυτό της εκτέλεσης, τα αλφαριθμητικά προστίθενται απ΄ ευθείας από το σώμα του κάθε μηνύματος στο συνολικό αλφαριθμητικό με χρήση της συνάρτησης `memcpy()` της βιβλιοθήκης της C.

Υλοποιώντας την παραπάνω ιδέα, προστέθηκε στη δομή κάθε κλειδιού ένας δείκτης σε χαρακτήρες (`char *hmigp`), που χρησιμοποιείται για να δείχνει στο αλφαριθμητικό όπου αποθηκεύεται η συνολική πληροφορία μετανάστευσης που αφορά το συγκεκριμένο κλειδί (φράγμα). Η δέσμευση της μνήμης για το αλφαριθμητικό αυτό γίνεται στη συνάρτηση αρχικοποίησης όλων των δομών που σχετίζονται με το συγχρονισμό στο JIAJIA , την `void initsyn()` και με χρήση της

[1] Αντίθετα μπορούν να έλθουν από διαφορετικούς σταθμούς παραπάνω από μία ειδοποιήσεις εγγραφής για την ίδια σελίδα και έτσι απαιτείται πιο πολύπλοκη δομή για την αποθήκευση των ειδοποιήσεων εγγραφής.

συνάρτησης `malloc()` της βιβλιοθήκης της C. Το μέγεθος της δεσμευόμενης μνήμης είναι πάντοτε `Maxmsgsize` bytes (αφού η πληροφορία αυτή θα μεταδοθεί στη συνέχεια στο σώμα του μηνύματος παραχώρησης του φράγματος, του οποίου το μέγιστο μέγεθος δεν ξεπερνά, όπως αναλύθηκε νωρίτερα). Η σπατάλη μνήμης είναι αμελητέα για να γίνει λόγος για μεταβαλλόμενα μεγέθη δεσμευόμενης μνήμης. Η απελευθέρωση της μνήμης που δεσμεύτηκε (στο διαχειριστή του φράγματος και μόνο) γίνεται όταν γίνεται και η εκκαθάριση των ειδοποιήσεων εγγραφής, με χρήση της συνάρτησης `free()` της βιβλιοθήκης της C, οπότε και γίνεται εκ νέου δέσμευση.

Προκειμένου να γίνεται σωστά η αντιγραφή των πληροφοριών μετανάστευσης του κάθε σταθμού στο αλφαριθμητικό όπου φυλάσσονται οι συνολικές πληροφορίες μετανάστευσης για κάποιο φράγμα, απαιτείται να είναι γνωστό το μέγεθος των πληροφοριών μετανάστευσης κάθε μηνύματος. Αυτό υπολογίζεται εύκολα, αφαιρώντας από το συνολικό μέγεθος του μηνύματος το μέγεθος όλων των άλλων δεδομένων του και αποθηκεύεται σε μία τοπική μεταβλητή του `barrserver()`, την `int hmigMsgInfo_size`. Το μήκος των δεδομένων του μηνύματος πλην των πληροφοριών μετανάστευσης μετράται από την ανάλυση του σώματος του μηνύματος μέχρι να βρεθεί ο διαχωριστής.

Προκειμένου να γίνεται σωστά η προσάρτηση των συνολικών πληροφοριών μετανάστευσης στο σώμα του μηνύματος παραχώρησης του φράγματος, η ανάλυση του σώματος του μηνύματος αυτού, αλλά και η συνένωση των επιμέρους πληροφοριών απαιτείται να είναι γνωστό και το συνολικό μέγεθος των πληροφοριών μετανάστευσης για κάθε συγκεκριμένο φράγμα. Η πληροφορία αυτή χρησιμοποιείται εκτός από τον `barrserver()` και στην `grantbarr()` και αναφέρεται σε κάθε φράγμα ξεχωριστά. Έτσι προστέθηκε ένας ακέραιος στη δομή κάθε κλειδιού (`int hmigInfo_size`). Ο αριθμός αυτός υπολογίζεται προσθέτοντας τα επιμέρους μεγέθη των πληροφοριών μετανάστευσης των μηνυμάτων. Η αρχικοποίησή του μεγέθους αυτού σε 0 γίνεται μαζί με την αρχικοποίηση όλων

των δομών συγχρονισμού, ενώ η επαναρχικοποίησή του γίνεται όταν καθαρίζονται και οι ειδοποιήσεις εγγραφής.

Τα πεδία που προσθέτουμε στη δομή του κλειδιού χρησιμοποιούνται βέβαια στα φράγματα («κρυφά» κλειδιά), αλλά μένουν αχρησιμοποίητα στα πραγματικά κλειδιά, που δεν αποτελούν σημεία λήψης μεταναστευτικών αποφάσεων. Ωστόσο η σπατάλη μνήμης είναι πολύ μικρή, μιας και στη δομή ορίζεται εκτός από τον ακέραιο `hmigInfo_size` μόνο ο δείκτης στο αλφαριθμητικό στο οποίο αποθηκεύονται οι πληροφορίες μετανάστευσης και όχι το ίδιο το αλφαριθμητικό.

3.4.9 Συλλογή όλων των πληροφοριών μετανάστευσης σελίδων μνήμης από το διαχειριστή φράγματος

Για τους λόγους που αναλύθηκαν, ο εκάστοτε διαχειριστής φράγματος είναι επιφορτισμένος με τη συλλογή όλων (από όλους τους σταθμούς) των πληροφοριών μετανάστευσης και τη μετάδοσή τους. Η συλλογή γίνεται στον `barrserver()`, αναλύοντας τα μηνύματα αίτησης φράγματος και εξάγοντας από το σώμα τους την πληροφορία μετανάστευσης, η οποία βρίσκεται αμέσως μετά τον ειδικό διαχωριστή (`STOP`). Η ανακάλυψη του διαχωριστή στο σώμα του μηνύματος γίνεται χρησιμοποιώντας τη συνάρτηση `strncmp()` της βιβλιοθήκης της `C`, που επιτρέπει σύγκριση συγκεκριμένου αριθμού χαρακτήρων κάποιων αλφαριθμητικών. Η επεξεργασία και η αποθήκευση των πληροφοριών ως αλφαριθμητικά και η εύρεση και αποθήκευση του συνολικού και των επιμέρους μεγεθών τους για κάθε μήνυμα γίνονται αθροιστικά, όπως περιγράφηκε παραπάνω. Η διαδικασία της συλλογής γίνεται φυσικά μόνο όταν είναι ενεργοποιημένος ο μηχανισμός μετανάστευσης σελίδων.

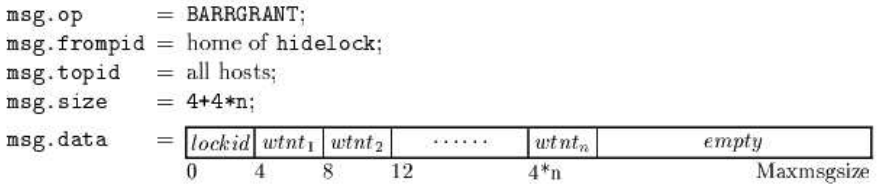
Τη συλλογή των πληροφοριών μετανάστευσης ακολουθεί η συνήθης διαδικασία καταγραφής (δηλαδή αντιστοίχησης με το συγκεκριμένο κλειδί ή φράγμα) των ειδοποιήσεων εγγραφής που έχουν φορτωθεί στο μήνυμα αίτησης

φράγματος. Η διαδικασία διαφοροποιείται κάπως όταν είναι ενεργοποιημένη η μετανάστευση σελίδων, μιας και η δομή των μηνυμάτων αίτησης φράγματος είναι διαφοροποιημένη. Έτσι η συνάρτηση καταγραφής των ειδοποιήσεων εγγραφής `void recordwtnts(jia_msg_t *req)` του JIAJIA που καλεί κατάλληλα τη συνάρτηση αποθήκευσης των ειδοποιήσεων εγγραφής `void savewtnt(wtnt_t *ptr, address_t addr, int frompid)` τροποποιήθηκε, ώστε να χειρίζεται με διαφορετικό τρόπο τα μηνύματα αίτησης φράγματος από άλλα μηνύματα που φέρουν ειδοποιήσεις εγγραφής, όταν βέβαια είναι ενεργοποιημένη η μετανάστευση σελίδων. Η διαφορά χειρισμού έγκειται στο γεγονός ότι η διαδικασία ανάλυσης του σώματος του μηνύματος για ειδοποιήσεις εγγραφής δε συνεχίζεται μέχρι το τέλος του μηνύματος, αλλά μέχρι να βρεθεί ο διαχωριστής `STOP`, αφού ακολουθούν οι πληροφορίες μετανάστευσης σελίδων. Η λύση του ξεχωριστού χειρισμού των μηνυμάτων με πληροφορία μετανάστευσης είναι η απλούστερη, αφού κοινή αντιμετώπιση όλων των μηνυμάτων θα απαιτούσε την εισαγωγή άχρηστου κατά τα άλλα διαχωριστή στο σώμα όλων των μηνυμάτων που περιέχουν ειδοποιήσεις εγγραφής.

Σημαντική είναι η παρατήρηση ότι ο διαχειριστής φράγματος συγκεντρώνει όλες τις πληροφορίες μετανάστευσης σελίδων, ακόμη και αυτές που αφορούν τον ίδιο. Αυτό συμβαίνει γιατί και ο ίδιος όπως και όλοι οι υπόλοιποι σταθμοί στέλνει μήνυμα αίτησης φράγματος (στον εαυτό του), το οποίο και αναλύει με την ίδια διαδικασία όπως και των υπόλοιπων. Τη συνολική πληροφορία μετανάστευσης σελίδων επίσης τη λαμβάνει όπως όλοι οι υπόλοιποι σταθμοί από το μήνυμα παραχώρησης φράγματος που στέλνει ο ίδιος (στον εαυτό του).

3.4.10 Μετάδοση όλων των πληροφοριών μετανάστευσης σελίδων μνήμης από το διαχειριστή φράγματος

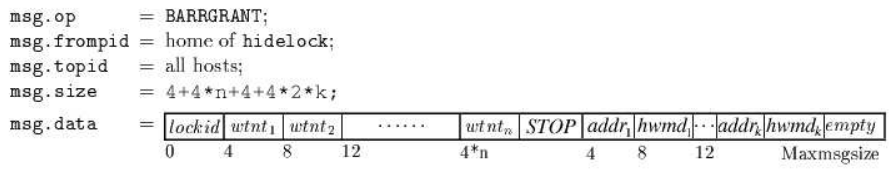
Αφού ο εκάστοτε διαχειριστής φράγματος συγκεντρώσει τις πληροφορίες μετανάστευσης από όλους τους σταθμούς έχει στη συνέχεια καθήκον να τις μεταδώ-



Σχήμα 3.4: Η αρχική δομή των μηνυμάτων τύπου BARRGRANT [13].

σει σε όλους. Με σκοπό τη μείωση του επικοινωνιακού κόστους η συνολική πληροφορία μετανάστευσης, όπως εξηγήθηκε, φορτώνεται στο σώμα του ήδη υπάρχοντος μήνυματος απόκρισης (παραχώρησης) φράγματος. Η φόρτωση λαμβάνει χώρα στο σώμα της συνάρτησης του JIAJIA που δημιουργεί το μήνυμα παραχώρησης φράγματος, δηλαδή της (`void grantbarr(long lock)`). Το μήνυμα παραχώρησης φράγματος του JIAJIA έχει κανονικά τη δομή που δείχνει το σχήμα 3.4. Παρατηρούμε ότι εκτός από τον αναγνωριστή του φράγματος που παραχωρείται περιλαμβάνονται και όσες ειδοποιήσεις εγγραφής χωράνε — και υπάρχουν.

Οι τροποποιήσεις στη δομή του μηνύματος παραχώρησης φράγματος, προκειμένου να φορτώνονται και οι πληροφορίες μετανάστευσης σελίδων, είναι ανάλογες με εκείνες στη δομή του μηνύματος αίτησης φράγματος. Και πάλι, προκειμένου να γίνεται σωστά η ανάλυση (parsing) του σώματος στον παραλήπτη, διαχωρίζονται οι πληροφορίες μετανάστευσης από τις προηγούμενες πληροφορίες που περιέχονται στο σώμα του μηνύματος (όπως για παράδειγμα τις ειδοποιήσεις εγγραφής), με ένα αλφαριθμητικό (πίνακα χαρακτήρων) μεγέθους 4 bytes (`STOP`), το οποίο υποδηλώνει το τέλος των προηγούμενων πληροφοριών και την αρχή των πληροφοριών μετανάστευσης σελίδων. Ο διαχωριστής προστίθεται ακόμη και στην περίπτωση που δεν έχουμε πληροφορίες μετανάστευσης σελίδων να φορτώσουμε στο μήνυμα (προκειμένου να ενοποιηθεί η διαδικασία ανάλυσης του μηνύματος από τον παραλήπτη), αλλά μόνο όταν είναι ενεργοποιημένη η μετανάστευση σελίδων μνήμης.



Σχήμα 3.5: Η τροποποιημένη δομή των μηνυμάτων τύπου BARRGRANT.

Αμέσως μετά τους τέσσερις χαρακτήρες του διαχωριστή ακολουθούν οι πληροφορίες μετανάστευσης. Πρόκειται στην παρούσα έκδοση και πολιτική μετανάστευσης, για τις διευθύνσεις μνήμης ισχυρά τροποποιημένων εντός έδρας σελίδων (4 bytes) και τους αντίστοιχους αναγνωριστές των ισχυρότερων —εξωτερικών βέβαια— τροποποιητών των σελίδων αυτών (επίσης 4 bytes). Όπως εξηγήθηκε, οι πληροφορίες αυτές για όλους τους σταθμούς είναι συνολικά αποθηκευμένες για κάθε φράγμα σε ένα αλφαριθμητικό, στο οποίο δείχνει ένας δείκτης στη δομή του φράγματος. Η δομή του τροποποιημένου μηνύματος αίτησης φράγματος δείχνεται στο σχήμα 3.5

Η προσάρτηση των πληροφοριών μετανάστευσης στο σώμα του μηνύματος απόκρισης φράγματος γίνεται και πάλι μέσω της συνάρτησης `void appendmsg(jia_msg_t *msg, unsigned char *str, int len)` του JIAJIA , χρησιμοποιώντας το μέγεθος του αλφαριθμητικού των συνολικών πληροφοριών μετανάστευσης που είναι αποθηκευμένο σε έναν ακέραιο στη δομή κάθε φράγματος, όπως εξηγήθηκε. Ακόμη και κατά τη μεταφορά της συνολικής πληροφορίας μετανάστευσης δεν ξεπεράστηκε το όριο μεγέθους μηνυμάτων του JIAJIA και έτσι δε χρειάστηκε να δημιουργηθεί επιπλέον τύπος μηνύματος, ειδικά για τη μετάδοση της πληροφορίας μετανάστευσης.

Με τον παραπάνω τρόπο ο διαχειριστής φράγματος ενημερώνει κάθε σταθμό για τις σελίδες όλων των σταθμών που τροποποιήθηκαν ισχυρά στο διάστημα που αφορά το συγκεκριμένο φράγμα, καθώς και από ποιο σταθμό έγιναν οι αντίστοιχες τροποποιήσεις, ώστε οι σελίδες αυτές να μεταναστεύσουν στους

αντίστοιχους σταθμούς. Όπως ήδη αναφέρθηκε, ο διαχειριστής φράγματος λαμβάνει και ο ίδιος, όπως όλοι οι υπόλοιποι σταθμοί, τη συνολική πληροφορία μετανάστευσης σελίδων, από το μήνυμα παραχώρησης φράγματος που στέλνει ο ίδιος (στον εαυτό του).

3.4.11 Ανάλυση όλων των πληροφοριών μετανάστευσης σελίδων μνήμης από όλους τους σταθμούς

Όπως ήδη αναφέρθηκε το JIAJIA προσφέρει μόνο καθολικά φράγματα. Έτσι σε ένα φράγμα κάθε σταθμός έχει στείλει μια αίτηση φράγματος, με φορτωμένες τις πληροφορίες μετανάστευσης σελίδων που εδράζονται σε αυτόν. Συνεπώς κάθε σταθμός περιμένει την απόκριση φράγματος, στην οποία ο διαχειριστής φράγματος έχει φορτώσει εκτός από τις ειδοποιήσεις εγγραφής και τις συνολικές πληροφορίες μετανάστευσης σελίδων για όλους τους σταθμούς. Η λήψη του μηνύματος απόκρισης (παραχώρησης) φράγματος γίνεται από τον αντίστοιχο εξυπηρέτη κάθε σταθμού, δηλαδή τον `void barrgrantserver(jia_msg_t *req)`. Αυτός αναλαμβάνει να καλέσει τη συνάρτηση `void invalidate(jia_msg_t *req)` του JIAJIA , η οποία αναλύει τις πληροφορίες που περιέχονται στο σώμα του μηνύματος παραχώρησης φράγματος και εκτελεί τις κατάλληλες ενέργειες ακύρωσης και μετανάστευσης σελίδων.

Όπως και στην περίπτωση χειρισμού των μηνυμάτων αίτησης φράγματος, η ανάλυση του σώματος των μηνυμάτων παραχώρησης φράγματος διαφοροποιείται κάπως όταν είναι ενεργοποιημένος ο μηχανισμός μετανάστευσης σελίδων. Αυτό συμβαίνει λόγω των τροποποιήσεων στη δομή των μηνυμάτων παραχώρησης φράγματος. Έτσι η `invalidate()` τροποποιήθηκε, ώστε να χειρίζεται ξεχωριστά τις περιπτώσεις μηνυμάτων παραχώρησης φράγματος, όταν βέβαια είναι ενεργοποιημένη η μετανάστευση σελίδων. Συγκεκριμένα, αντί η διαδικασία

ανάλυσης του σώματος του μηνύματος για ειδοποιήσεις εγγραφής και ακύρωσης των κατάλληλων σελίδων της λανθάνουσας μνήμης να συνεχίζεται μέχρι το τέλος του μηνύματος, συνεχίζεται μέχρι να βρεθεί ο διαχωριστής `STOP` στο σώμα (ο οποίος ανακαλύπτεται πάλι με χρήση της `strncmp()`). Η λύση του ξεχωριστού χειρισμού των μηνυμάτων με πληροφορία μετανάστευσης, αν και έχει το μειονέκτημα της επανάληψης ενός (μικρού) τμήματος κώδικα, είναι η απλούστερη, μιας και δεν επιβάλλει αλλαγές στους άλλους τύπους μηνυμάτων, ούτε περιπλέκει τον αλγόριθμο ανάλυσης του σώματος.

Μέχρι να συναντήσει το διαχωριστή, κάθε σταθμός που έχει λάβει μια απόκριση φράγματος ελέγχει κάθε ληφθείσα ειδοποίηση εγγραφής και το αντίστοιχο πεδίο τροποποιητή. Εάν η ειδοποίηση εγγραφής και το αντίστοιχο πεδίο τροποποιητή δείχνουν πως η σελίδα τροποποιήθηκε κατά τη διάρκεια του τελευταίου διαστήματος μεταξύ φραγμάτων από πολλαπλούς σταθμούς ή από ένα σταθμό διαφορετικό από τον παρόντα, τότε η αντίστοιχη σελίδα ακυρώνεται εάν φυλάσσονταν στη λανθάνουσα μνήμη. Στη συνέχεια ελέγχεται το λιγότερο σημαντικό bit της διεύθυνσης που περιέχει η ειδοποίηση εγγραφής (`migtag`). Εάν το bit αυτό είναι «1», τότε η λανθάνουσα μνήμη του σταθμού που έστειλε την ειδοποίηση εγγραφής έχει ένα έγκυρο αντίγραφο της σελίδας. Αυτό εξασφαλίζεται από τον τρόπο δημιουργίας των ειδοποιήσεων εγγραφής στο JIAJIA και συμβαίνει στην περίπτωση που ο σταθμός που τροποποίησε τη σελίδα ήταν ο μοναδικός τροποποιητής. Αυτές ήταν και οι μοναδικές σελίδες που μετανάστευαν με το υπάρχον πρωτόκολλο και η μετανάστευσή τους δεν απαιτούσε μετακίνηση της σελίδας, αφού αυτή υπήρχε ήδη στη λανθάνουσα μνήμη της νέας έδρας.[2] Ακόμη και με το νέο πρωτόκολλο —και για λόγους αύξησης της ταχύτητας απόκρισης και εξοικονόμησης εύρους ζώνης— δεν υπάρχει λόγος να

[2]Σε μία διαφορετική υλοποίηση θα μπορούσαν και να μη στελνονται καθόλου πληροφορίες μετανάστευσης για τις σελίδες αυτές, μειώνοντας έτσι λίγο το επικοινωνιακό κόστος, εις βάρος όμως της απλότητας υλοποίησης και του κοινού τρόπου διαχείρισης των μεταναστεύσεων. Το τελευταίο είναι ιδιαίτερα σημαντικό και για την εφαρμογή διαφορετικών πολιτικών μετανάστευσης.

ζητήσει η νέα έδρα κάποια τέτοια σελίδα από την παλαιά έδρα, εφόσον έχει ήδη έγκυρο αντίγραφο στη λανθάνουσα μνήμη της. Συνεπώς οι διευθύνσεις των σελίδων των οποίων υπάρχει έγκυρο αντίγραφο στη λανθάνουσα μνήμη κάποιου σταθμού σημειώνονται ξεχωριστά σε έναν τοπικό —εντός της `invalidate()`— πίνακα, ο οποίος ονομάστηκε `address_t validInCache[[Cachepages+1]` και έχει σταθερό, επαρκές μέγεθος, χωρίς να σπαταλάται πολλή μνήμη.

Το διαχωριστή ακολουθούν στο σώμα του μηνύματος (και μέχρι το τέλος αυτού) οι πληροφορίες μετανάστευσης σελίδων για το συγκεκριμένο φράγμα, για σελίδες που εδράζονται σε όλους τους σταθμούς. Πρόκειται για τις διευθύνσεις μνήμης των ισχυρά τροποποιημένων από εξωτερικούς σταθμούς σελίδων, καθώς και για τους αναγνωριστές των τροποποιητών αυτών, οι οποίοι θα γίνουν και οι νέες έδρες των αντίστοιχων σελίδων μνήμης. Όλες οι πληροφορίες είναι απαραίτητες σε όλους τους σταθμούς, προκειμένου να ανανεώσουν κατάλληλα τον καθολικό πίνακα σελίδων. Επιπλέον ενέργειες λαμβάνουν χώρα στην παλαιά και στη νέα έδρα κάθε μεταναστεύουσας σελίδας.

Η συνάρτηση που καλείται από την `invalidate()`, προκειμένου να ολοκληρώσει τη διαδικασία της μετανάστευσης ενημερώνοντας όλους τους πίνακες σελίδων της κοινής μνήμης είναι η `void migpage(unsigned long addr,int frompid,int topid)`[3]. Με το υπάρχον πρωτόκολλο η συνάρτηση αυτή καλούνταν μόνο για σελίδες που βρίσκονταν σε έγκυρη κατάσταση στη λανθάνουσα μνήμη κάποιου σταθμού, ο οποίος ήταν και ο μοναδικός τροποποιητής τους. Μιας και με το νέο πρωτόκολλο αυτή δεν είναι η μοναδική περίπτωση μετανάστευσης σελίδων, προστέθηκε, για να τη διαχωρίζει, ένα επιπλέον όρισμα στη συνάρτηση `migpage()`, το `int singlemodifier`, το οποίο λειτουργεί ως σημαία. Η διεύθυνση κάθε σελίδας προς μετανάστευση ελέγχεται αν περιλαμβάνεται στον πίνακα `validInCache[]`, οπότε και η `migpage()` καλείται με `singlemodifier`

[3]Δύο ακόμη συναρτήσεις που προϋπήρχαν στο JIAJIA και αφορούν τη μετανάστευση σελίδων είναι η `void migarrangehome()` και η `void migcheckcache()`, για την αναδιοργάνωση ύστερα από μεταναστεύσεις των πινάκων σελίδων εδρών και λανθάνουσας μνήμης αντίστοιχα. Οι συναρτήσεις αυτές χρησιμοποιήθηκαν ως είχαν.

«1», ειδάλλως με «0». Κάθε σταθμός καλεί τη συνάρτηση migpage() για όλες τις σελίδες που μεταναστεύουν. Οι ενέργειες της συνάρτησης αυτής διαφοροποιούνται ανάλογα με τη σχέση του σταθμού με τη μεταναστεύουσα σελίδα.

3.4.12 Ενημέρωση των πινάκων σελίδων της κοινής μνήμης

Ενέργειες στη νέα έδρα

Εάν ο σταθμός είναι η νέα έδρα της μεταναστεύουσας σελίδας, και ήταν και ο μοναδικός τροποποιητής της και έχει ήδη ένα έγκυρο αντίγραφό της στη λανθάνουσα μνήμη του, δε χρειάζεται (ούτε και είναι σωστό) να τη ζητήσει από την έως τότε έδρα. Εάν ο σταθμός είναι η νέα έδρα της μεταναστεύουσας σελίδας, αλλά όχι ο μοναδικός τροποποιητής της, δεν έχει έγκυρο αντίγραφο της σελίδας. Συνεπώς πρέπει να ζητήσει τη σελίδα από την έως τότε έδρα της, η οποία είναι και η μόνη που έχει έγκυρο αντίγραφο, αφού ως έδρα της σελίδας είναι ο αποδέκτης όλων των διαφορών που την αφορούν.[4] Η λήψη της σελίδας γίνεται χρησιμοποιώντας τη συνάρτηση void getpage(address_t addr,int flag). Η συνάρτηση αυτή δημιουργεί ένα μήνυμα αίτησης λήψης σελίδας (GETP), το οποίο στέλνει στη μέχρι τότε έδρα της σελίδας. Στη συνέχεια ο αντίστοιχος εξυπηρέτης (void getpserver(jia_msg_t *req)), που βρίσκεται στην έδρα της σελίδας που ζητείται, παίρνει τη διεύθυνση της ζητούμενης σελίδας από το μήνυμα της αίτησης, διαβάζει τη σελίδα από τη μνήμη και τη στέλνει στο σταθμό που τη ζήτησε, στο μήνυμα παραχώρησης λήψης σελίδας (GETPGRANT). Ο σταθμός εκείνος την παραλαμβάνει με τη σειρά του μέσω του αντίστοιχου εξυπηρέτη (void getpgrantserver(jia_msg_t *rep)). Ο μηχανισμός αυτός

[4] Οι διαφορές δε θα μπορούσαν να μεταδίδονται απευθείας προς τη νέα έδρα, αφού αυτή γνωστοποιείται σε όλους τους σταθμούς αργότερα.

προϋπήρχε στο JIAJIA , για την εξυπηρέτηση των σημάτων τύπου SIGSEGV για σελίδες εκτός έδρας. Έτσι η σημαία flag της getpage() δεχόταν δύο διαφορετικές τιμές, «0» για αιτήσεις ανάγνωσης και «1» για αιτήσεις εγγραφής. Η getpage() τοποθετούσε την εκάστοτε τιμή στο πεδίο temp του μηνύματος αίτησης λήψης σελίδας, ώστε και ο getpserver() να ενεργεί ανάλογα, μιας και στην περίπτωση των αιτήσεων εγγραφής και εφόσον είναι ενεργοποιημένο το διάνυσμα εγγραφών δεν απαιτείται η αποστολή όλης της σελίδας. Στη σημαία αυτή προστέθηκε μία ακόμη δυνατή τιμή, «2», για αιτήσεις μετανάστευσης σελίδων.

Στη συνέχεια ενημερώνονται οι πίνακες σελίδων για την αλλαγή. Στον πίνακα σελίδων που εδράζονται στο σταθμό δίνεται μια θέση για τη νεοφερμένη σελίδα. Μιας και η δομή της σελίδας δημιουργείται εκ νέου, μηδενίζονται και όλες οι μετρήσεις διαφορών που αφορούσαν την προηγούμενη έδρα, γεγονός επιθυμητό. Σε μία πιο πολύπλοκη πολιτική μετανάστευσης θα μπορούσαν να μεταφέρονται και επιπλέον πληροφορίες που θα αποθηκεύονταν στη νέα δομή. Αφού η σελίδα αποθηκεύεται πλέον στις εντός έδρας σελίδες του σταθμού, η θέση της στον πίνακα των σελίδων που κρατούνται στη λανθάνουσα μνήμη αποδεσμεύεται. Τέλος το πεδίο που αναφέρει το σταθμό-έδρα της σελίδας αυτής στην αντίστοιχη θέση του καθολικού πίνακα σελίδων τίθεται να δείχνει στη νέα έδρα.

Ενέργειες στην παλαιά έδρα

Εάν ο σταθμός είναι η παλαιά έδρα της μεταναστεύουσας σελίδας, τότε η αντίστοιχη θέση της στον πίνακα των σελίδων που εδράζονται στο σταθμό αυτό αποδεσμεύεται, η σελίδα ακυρώνεται εάν δεν υπάρχει ελεύθερη θέση στον πίνακα των σελίδων που κρατούνται στη λανθάνουσα μνήμη ή κρατείται ως έγκυρη εάν βρεθεί ελεύθερη θέση για τη σελίδα αυτή στον πίνακα αυτό και το πεδίο

που αναφέρει το σταθμό-έδρα της σελίδας αυτής στην αντίστοιχη θέση του καθολικού πίνακα σελίδων τίθεται να δείχνει στη νέα έδρα.

Ενέργειες σε κάθε άλλο σταθμό

Εάν ο σταθμός δεν είναι ούτε η νέα ούτε η παλαιά έδρα της μεταναστεύουσας σελίδας, τότε μόνο το πεδίο που αναφέρει το σταθμό-έδρα της σελίδας αυτής στην αντίστοιχη θέση του καθολικού πίνακα σελίδων τίθεται να δείχνει στη νέα έδρα.

3.4.13 Απαραίτητοι συγχρονισμοί

Στην περίπτωση που η νέα έδρα της μεταναστεύουσας σελίδας χρειάζεται να ζητήσει ένα έγκυρο αντίγραφό της από την παλαιά έδρα, δημιουργούνται επιπλέον ανάγκες συγχρονισμού. Συγκεκριμένα, πρέπει να εξασφαλισθεί ότι:

1. Η νέα έδρα δε θα αναφερθεί στη νεοαποκτηθείσα σελίδα, πριν ακόμη αυτή έλθει πραγματικά από την παλαιά έδρα.
2. Η παλαιά έδρα δε θα αλλάξει τους πίνακες σελίδων της και δε θα αποαντιστοιχίσει τη σελίδα, πριν τη στείλει στη νέα έδρα.

3.4.14 Μεταβλητές συγχρονισμού

Το JIAJIA διαθέτει δύο μεταβλητές για το συγχρονισμό της αποστολής διαφορών και της παραλαβής και εφαρμογής τους και για το συγχρονισμό της

ζήτησης σελίδων και της παραλαβής τους. Πρόκειται για τις `volatile int diffwait` και `volatile int getpwait` αντίστοιχα. Η `getpwait` μπορεί να βοηθήσει στην εξασφάλιση της πρώτης συνθήκης. Η μεταβλητή αυτή αρχικοποιείται σε «0» στην `initmem()`, τίθεται σε «1» από την `getpage()` όταν ζητείται μία σελίδα και επανατίθεται σε «0» από τον `getpgrantserver()` όταν έχει παραχωρηθεί πλέον η σελίδα. Συνεπώς μία αναμονή μέχρι η `getpwait` να γίνει «0», (`while(getpwait);`) μπορεί να εξασφαλίσει ότι μία σελίδα που ζητήθηκε έχει παραχωρηθεί από την έδρα της, οπότε και μπορούν πλέον να γίνουν αναφορές σε αυτή.

Προκειμένου να εξασφαλισθεί η δεύτερη συνθήκη, κρίθηκε απαραίτητη η εισαγωγή μίας ακόμη μεταβλητής συγχρονισμού, της `volatile int hmigwait` (ορισμένης ως volatile, μιας και η τιμή της τίθεται πιθανά από πολλές διαφορετικές διεργασίες). Η μεταβλητή αυτή αρχικοποιείται σε «0» στην `initmem()`, όπως και οι υπόλοιπες. Αυξάνεται κατά 1 από κάθε σταθμό, για κάθε εντός έδρας σελίδα για την οποία στέλνει πληροφορίες μετανάστευσης (και συνεπώς πρόκειται να μεταναστεύσει). Μειώνεται δε κατά 1 από κάθε σταθμό, για κάθε εντός έδρας σελίδα την οποία στέλνει σε κάποιον άλλο σταθμό μέσω του `getpserver()`, εφόσον βέβαια πρόκειται για απάντηση σε αίτηση μετανάστευσης σελίδας. [5] Εφόσον υπάρχουν και οι σελίδες που μεταναστεύουν κατευθείαν στο μοναδικό τροποποιητή τους χωρίς να ζητηθούν από την έδρα τους, η μεταβλητή `hmigwait` μειώνεται κατά 1 και από την `invalidate()`, για κάθε σελίδα η διεύθυνση της οποίας υπάρχει στον πίνακα `validInCache[]`. Με τα παραπάνω και ελέγχοντας πότε η `hmigwait` γίνεται «0», εξασφαλίζεται ότι οι αλλαγές στους πίνακες σελίδων της παλαιάς έδρας μιας σελίδας και η πιθανή αποαντιστοίχιση κάποιας μεταναστεύουσας σελίδας δε θα λάβουν χώρα πριν η σελίδα αυτή σταλεί στη νέα έδρα της, εφόσον βέβαια κάτι τέτοιο χρειάζεται.

[5] Για το λόγο αυτό κρίθηκε απαραίτητη και η διαφοροποίηση των αιτήσεων μετανάστευσης σελίδας από τις αιτήσεις για ανάγνωση ή εγγραφή, μέσω μιας ακόμη τιμής στη σημαία `flag`.

3.4.15 Σημεία ελέγχου των μεταβλητών συγχρονισμού

Είναι σημαντικό οι έλεγχοι για το μηδενισμό τόσο της getpwait όσο και της hmigwait να γίνονται στο σωστό σημείο του κώδικα ώστε να αποφευχθούν τα αδιέξοδα. Ένας απλός έλεγχος εντός της migpage() σίγουρα δεν αποτελεί λύση. Ο βασικός λόγος για αυτό είναι ο εξής: Το JIAJIA δέχεται όλα τα σήματα τύπου SIGIO ασύγχρονα (εφόσον δεν είναι απενεργοποιημένη η λήψη τους), αλλά εκτελεί τους αντίστοιχους εξυπηρέτες διαδοχικά (σύγχρονα). Έτσι λαμβάνοντας ένας σταθμός ένα μήνυμα τύπου BARRGRANT εκκινεί τον αντίστοιχο εξυπηρέτη, ο οποίος καλεί την invalidate() και αυτή με τη σειρά της τη migpage(). Μία αναμονή εντός της migpage μέχρι η hmigwait να γίνει μηδέν θα ήταν αέναη, αφού οι αιτήσεις για σελίδες (μηνύματα τύπου GETP) περιμένουν να εξυπηρετηθούν από τον αντίστοιχο εξυπηρέτη, όταν τελειώσει η εκτέλεση του προηγούμενου εξυπηρέτη, ο οποίος περιμένει αυτές! Προφανώς απαιτείται είτε η διακοπτόμενη εκτέλεση των εξυπηρετών, είτε η εξυπηρέτηση των αιτήσεων σελίδων πριν την εξυπηρέτηση των μηνυμάτων παραχώρησης φράγματος.

Μία ασύγχρονη (διαπλεκόμενη) εκτέλεση των εξυπηρετών θα συνεπάγονταν αλλαγή της φιλοσοφίας του χειρισμού μηνυμάτων του JIAJIA , που θα οδηγούσε πιθανά και σε νέα προβλήματα διαπλοκής και αδιέξοδα. Έτσι απορρίπτεται η περίπτωση να διακόπτεται προσωρινά η εκτέλεση του barrgrantserver() για να εκτελεσθεί ο getpserver(). Ούτως η άλλως η υλοποίηση θα ήταν ιδιαίτερα πολύπλοκη, με διαδοχικές κλήσεις μεταξύ των εξυπηρετών. Επιπλέον δε διαφαίνεται κάποιο ιδιαίτερο πλεονέκτημα αποδοτικότητας, αφού θα πρέπει ούτως ή άλλως να γίνουν όλες οι ενημερώσεις και αποαντιστοιχίσεις πριν τελειώσει το φράγμα, ενώ αυξάνονται και οι μεταβάσεις από τον κώδικα που αφορά τις ενημερώσεις στον κώδικα που αφορά τις εξυπηρετήσεις και αντίστροφα.

Μία άλλη προσέγγιση θα ήταν η κανονική εκτέλεση του barrgrantserver(), αλλά με τροποποιημένη τη migpage() ώστε να μη γίνεται στην παρούσα φάση

η ενημέρωση των πινάκων σελίδων και η αποαντιστοίχιση των σελίδων που πρόκειται να σταλούν. Αυτή θα έπρεπε να γίνεται ξεχωριστά αργότερα, όταν θα έχουν εξυπηρετηθεί όλες οι αιτήσεις λήψης σελίδων, πριν βέβαια το τέλος του φράγματος. Η διάσπαση του κώδικα της `migpage()` όμως επιτείνει την πολυπλοκότητα, ενώ δεν αποφεύγεται η αναμονή.

Προτιμητέα κρίθηκε λοιπόν η λύση της αναμονής μέχρι την εξυπηρέτηση όλων των αιτήσεων λήψης σελίδων από τον `getpserver()`, πριν ξεκινήσει η εξυπηρέτηση του μηνύματος παραχώρησης φράγματος από τον `barrgrantserver()`, οπότε και μπορεί να γίνει κατευθείαν η ενημέρωση των πινάκων σελίδων και η αποαντιστοίχηση όλων των σελίδων. Δεν είναι απαραίτητο να διαχωρίζεται ποιες σελίδες έχουν αποσταλεί, αρκεί να έχουν αποσταλεί όλες. Αυτό διότι όλες οι εξυπηρετήσεις αφορούν το τρέχον κάθε φορά φράγμα, αφού προκειμένου να προχωρήσει κάποιος σταθμός σε άλλο φράγμα και να ζητήσει νέες σελίδες για μετανάστευση, θα πρέπει να έχουν φύγει όλοι οι σταθμοί από το τρέχον φράγμα.

Η λύση της αναμονής μέχρι την εξυπηρέτηση όλων των αιτήσεων λήψης σελίδων πριν την εξυπηρέτηση της παραχώρησης φράγματος, αν και η πιο απλή και αποδοτική, εισάγει ένα νέο ερώτημα: Πρέπει η αναμονή να εισαχθεί σε κατάλληλο σημείο, ώστε να μπορούν να εξυπηρετούνται οι αιτήσεις για σελίδες, αλλά και να μην εξυπηρετηθεί η παραχώρηση φράγματος νωρίτερα από ότι πρέπει. Συνεπώς το σημείο ελέγχου θα πρέπει να βρίσκεται εντός του χειριστή των σημάτων τύπου `SIGIO`. Το JIAJIA προσφέρει παραλαβή και εξυπηρέτηση των μηνυμάτων με τη σειρά που αυτά δημιουργήθηκαν, παρόλο που το UDP/IP δεν εξασφαλίζει κάτι τέτοιο. Αυτό επιτυγχάνεται με τη χρήση μηνυμάτων επιβεβαίωσης. Ο έλεγχος για σωστή σειρά παραλαβής όμως αφορά μηνύματα που δημιουργήθηκαν στον ίδιο σταθμό. Δεν υπάρχει —από την πλευρά της σειράς δημιουργίας μηνυμάτων— άμεσος ελέγχος της διαδοχής γεγονότων από διαφορετικούς σταθμούς. Συνεπώς είναι πιθανό ένας σταθμός να λάβει (και

άρα να εξυπηρετήσει) ένα μήνυμα παραχώρησης φράγματος πριν από κάποιο μήνυμα αίτησης λήψης σελίδας. Προκύπτει άρα η ανάγκη για *αναδιάταξη* των εισερχόμενων μηνυμάτων στην αντίστοιχη ουρά, ώστε οι εξυπηρέτες να εκτελούνται με την επιθυμητή σειρά. Συγκεκριμένα θα πρέπει κάθε φορά που καταφθάνει ένα μήνυμα τύπου BARRGRANT και εφόσον δεν έχουν εξυπηρετηθεί όλες οι αιτήσεις λήψης σελίδων (εφόσον δηλαδή η μεταβλητή hmigwait δεν είναι «0») το μήνυμα αυτό να τοποθετείται μετά από οποιοδήποτε μήνυμα τύπου GETP στην ουρά εισερχόμενων μηνυμάτων ή στο τέλος της ουράς αυτής. Για την εκτέλεση της εργασίας αυτής υλοποιήθηκε ένας προεξυπηρέτης μηνυμάτων (void msgpreserver()), ο οποίος καλείται στις παραπάνω περιπτώσεις πριν τον κανονικό εξυπηρέτη μηνυμάτων (void msgserver()). Σημειώνεται ότι δεν υπάρχει ο κίνδυνος της λιμοκτονίας (starvation) των μηνυμάτων τύπου BARRGRANT, αφού κάποια στιγμή όλοι οι σταθμοί περιμένουν την εξυπηρέτηση αυτών των μηνυμάτων για να προχωρήσουν στο επόμενο φράγμα.

Μία απλή υλοποίηση του προεξυπηρέτη μηνυμάτων, που τοποθετεί το μήνυμα BARRGRANT στο τέλος της ουράς εισερχόμενων μηνυμάτων, είναι η εξής:

```
void msgpreserver(){

  jia_msg_t tmp;
  int i;

  /* While accessing the queue, better be in a CS.  --repantis */
      BEGINCS;
      memcpy(&tmp, &(inqh), Msgheadsize+inqh.size);

      for(i=inhead; i<intail; i++)
        memcpy(&(inqueue[i]),&(inqueue[i+1]),Msgheadsize+inqueue[i+1].size);
```

```
      memcpy(&(inqueue[intail-1]), &tmp, Msgheadsize+tmp.size);
      ENDCS;
}
```

Η παραπάνω υλοποίηση αν και αποδοτική, μιας και η μετακίνηση είναι άμεση (αφού όλα τα μηνύματα μετακινούνται κατά μία θέση μπροστά, για να τοποθετηθεί το μήνυμα BARRGRANT στο τέλος) έχει το εξής μειονέκτημα: Πιθανά αλλάζει τη σειρά εξυπηρέτησης και άλλων γεγονότων, των οποίων τα μηνύματα τοποθετούνται έτσι ώστε να εξυπηρετηθούν νωρίτερα από το μήνυμα παραχώρησης φράγματος. Προκειμένου να αποφευχθεί αυτό χρησιμοποιήθηκε μία ακόμη υλοποίηση του προεξυπηρέτη μηνυμάτων, η ακόλουθη, η οποία τοποθετεί το μήνυμα BARRGRANT μετά από το τελευταίο μήνυμα GETP στην ουρά εισερχόμενων μηνυμάτων. Η υλοποίηση αυτή αλλάζει τη σειρά εξυπηρέτησης των ελάχιστων δυνατών μηνυμάτων. Έχει ωστόσο το μειονέκτημα της μεγαλύτερης πολυπλοκότητας (αφού πρέπει να βρεθεί η θέση του τελευταίου μηνύματος GETP και να μετακινηθούν τα προηγούμενα μηνύματα κατά μία θέση μπροστά, ώστε να τοποθετηθεί το μήνυμα BARRGRANT), που συνεπάγεται κάποια καθυστέρηση κατά την εκτέλεση.

```
void msgpreserver(){

  jia_msg_t tmp;
  int i;
  int newBGRANTposition=-1;
  /* The former position of the last GETP,
     initialized to an impossible position.  --repantis */

  /* While accessing the queue, better be in a CS.  --repantis */
```

```
      BEGINCS;
      memcpy(&tmp, &(inqh), Msgheadsize+inqh.size);

      /* Find the last GETP on the inqueue.  --repantis */
      for(i=intail-1; i>=inhead; i--)
        if(inqueue[i].op==GETP)
          newBGRANTposition = i;
        /* New BARRGRANT position is after the last GETP msg
           (which will be moved together with the other msgs
           one position forward).  --repantis */

     if(newBGRANTposition != -1){
      for(i=inhead; i<newBGRANTposition; i++)
        memcpy(&(inqueue[i]),&(inqueue[i+1]),Msgheadsize+inqueue[i+1].size);
      memcpy(&(inqueue[newBGRANTposition]), &tmp, Msgheadsize+tmp.size);
     }
      ENDCS;
}
```

Προκειμένου να επιβεβαιωθεί ότι η αναδιάταξη των μηνυμάτων της ουράς των εισερχομένων γίνεται σωστά, τυπώθηκαν σε αρκετές δοκιμές τα μηνύματα της ουράς πριν και μετά την αναδιάταξη. Για το σκοπό αυτό χρησιμοποιήθηκε η συνάρτηση `void printmsg(jia_msg_t *msg, int right)` του JIAJIA και μάλιστα τροποποιημένη, ώστε να τυπώνει όλο το σώμα με τα δεδομένα του μηνύματος.

Σε οποιαδήποτε από τις δύο υλοποιήσεις, οι μεταβολές στην ουρά περιέχονται εντός ενός κρίσιμου τομέα, κατά τον οποίο τα σήματα τύπου SIGIO είναι φραγμένα (blocked) ώστε να μην είναι επιτρεπτή η εξυπηρέτηση νέων σημάτων

αυτού του τύπου, που θα άλλαζε ξανά την κατάσταση στην ουρά εισερχομένων μηνυμάτων. Χρησιμοποιώντας τον προεξυπηρέτη μηνυμάτων πριν τον κανονικό εξυπηρέτη, εφόσον η μεταβλητή συγχρονισμού hmigwait δεν έχει μηδενισθεί ακόμη, εξασφαλίζεται ότι η παλαιά έδρα θα στείλει τις σελίδες που τις έχουν ζητηθεί για μετανάστευση πριν τις αποαντιστοιχίσει και αλλάξει τους πίνακες σελίδων της.

Η εξασφάλιση ότι η νέα έδρα δε θα αναφερθεί στις νεοαποκτηθείσες σελίδες πριν αυτές έλθουν πραγματικά γίνεται —όπως αναφέρθηκε— αναμένοντας να μηδενισθεί η μεταβλητή συγχρονισμού getpwait. Η αναμονή αυτή πρέπει να γίνεται σε τέτοιο σημείο του κώδικα που επίσης να μην προκαλεί αδιέξοδο. Και πάλι δεν είναι δυνατός ο έλεγχος της getpwait εντός της migpage(), αφού όπως εξηγήθηκε η εκτέλεση των εξυπηρετών στο JIAJIA γίνεται διαδοχικά. Επιπλέον δε θα ήταν δυνατή αντίστοιχη αναδιάταξη των μηνυμάτων BARRGRANT πριν τα μηνύματα GETPGRANT, αφού η διαδικασία ξεκινά με την κλήση της getpage(), η οποία γίνεται κατά την εξυπηρέτηση των μηνυμάτων BARRGRANT. Συνεπώς πρέπει τα μηνύματα GETPGRANT να εξυπηρετούνται όταν έρχονται, μετά δηλαδή από τα μηνύματα BARRGRANT. Οπότε οι αλλαγές στους πίνακες των σελίδων θα πρέπει να γίνονται μετά την εξυπηρέτηση των μηνυμάτων GETPGRANT. Εναλλακτικά, οι αλλαγές μπορούν να γίνουν κατά την εξυπηρέτηση των μηνυμάτων BARRGRANT, αλλά να ακολουθεί αναμονή για το μηδενισμό της getpwait, πριν ακόμη χρησιμοποιηθούν οι νέοι πίνακες σελίδων. Αυτή είναι και η απλούστερη λύση, μιας και δεν απαιτείται διάσπαση του κώδικα της migpage(). Συνεπώς επιλέχθηκε η αναμονή για το μηδενισμό της getpwait να γίνεται εντός του φράγματος (εντός της jia_barrier()), αλλά εκτός της εξυπηρέτησης των μηνυμάτων BARRGRANT (μετά από αυτήν), ώστε να είναι δυνατή η ασύγχρονη ανανέωση της τιμής της μεταβλητής από τον εξυπηρέτη παραχώρησης λήψης σελίδων. Έτσι παρόλο που έχουν γίνει οι αλλαγές στους πίνακες σελίδων οι νέες σελίδες εξασφαλίζεται ότι δε θα χρησιμοποιηθούν πριν έλθουν πραγματικά.

Προφανώς η χρήση του προεξυπηρέτη μηνυμάτων και ο έλεγχος για το μηδενισμό των hmigwait και getpwait γίνονται μόνο όταν είναι ενεργοποιημένος ο μηχανισμός μετανάστευσης σελίδων, οπότε και προκύπτει η ανάγκη για την εξασφάλιση των αντίστοιχων συγχρονισμών.

3.4.16 Βελτιωμένη αντιμετώπιση των αναγκών συγχρονισμού

Η παραπάνω αντιμετώπιση των αναγκών συγχρονισμού των κόμβων, αν και λειτουργεί, είναι επιρρεπής σε σφάλματα και αδιέξοδα λόγω της πολυπλοκότητάς της και της αναδιάταξης των εισερχόμενων μηνυμάτων. Υλοποιήθηκε έτσι μία ακόμη λύση, πιο απλή και στιβαρή, η οποία εισάγει όμως την καθυστέρηση ενός επιπλέον σημείου συγχρονισμού.

Συγκεκριμένα, όπως και με την προηγούμενη λύση, όλοι οι σταθμοί καλούν μέσα από την invalidate() τη συνάρτηση migpage() για όλες τις σελίδες που αναφέρεται στο μήνυμα παραχώρησης φράγματος ότι θα μεταναστεύσουν. Στη νέα υλοποίηση όμως, η migpage() δεν πραγματοποιεί[6] η ίδια τις αλλαγές στους πίνακες σελίδων, ούτε ζητάει σελίδες. Η μόνη περίπτωση που η migpage() αλλάζει κάποιον πίνακα σελίδων, είναι αυτή κατά την οποία ο σταθμός δεν είναι ούτε η νέα ούτε η παλαιά έδρα της μεταναστεύουσας σελίδας. Τότε η migpage() ανανεώνει το πεδίο που αναφέρει το σταθμό-έδρα της σελίδας αυτής στην αντίστοιχη θέση του καθολικού πίνακα σελίδων, ώστε να δείχνει στη νέα έδρα. Σε κάθε άλλη περίπτωση η migpage() αποθηκεύει τις πληροφορίες για τις μεταναστεύουσες σελίδες σε δύο διαφορετικούς πίνακες, των οποίων το μέγεθος ορίζεται στατικά για λόγους ταχύτητας και απλότητας και μιας και η σπατάλη μνήμης είναι αμελητέα. Σε κάθε φράγμα οι πίνακες

[6] Εφόσον εκτελείται σε σταθμό που συνδέεται άμεσα με τη μεταναστεύουσα σελίδα, είτε ως παλαιά είτε ως νέα έδρα της.

pagesArriving[][] και pagesLeaving[][] γεμίζουν με πληροφορίες για τις σελίδες που μεταναστεύουν σε αυτό.

Στην πρώτη στήλη κάθε γραμμής του διδιάστατου πίνακα pagesArriving[][] αποθηκεύεται η διεύθυνση μιας σελίδας που πρόκειται να μεταναστεύσει *στον* εκάστοτε σταθμό, ενώ στις επόμενες στήλες αποθηκεύονται επιπλέον πληροφορίες που την αφορούν. Στην παρούσα υλοποίηση υπάρχει μία ακόμη στήλη, στην οποία η migpage() αποθηκεύει την πληροφορία αν η σελίδα τροποποιήθηκε μόνο από έναν κόμβο (singlemodifier).

Στην πρώτη στήλη κάθε γραμμής του διδιάστατου πίνακα pagesLeaving[][] αποθηκεύεται η διεύθυνση μιας σελίδας που πρόκειται να μεταναστεύσει *από* τον εκάστοτε σταθμό, ενώ στις επόμενες στήλες αποθηκεύονται επιπλέον πληροφορίες για αυτήν. Στη συγκεκριμένη υλοποίηση αποθηκεύεται σε μία ακόμη στήλη ο αναγνωριστής του εκάστοτε σταθμού που πρόκειται να γίνει νέα έδρα της σελίδας.

Στην προηγούμενη υλοποίηση της μετανάστευσης σελίδων, η οποία είχε βασισθεί στην υπάρχουσα συνάρτηση migpage(), όλοι οι κόμβοι πραγματοποιούσαν τις αλλαγές που απαιτούνταν στους πίνακες σελίδων μπλεγμένα, ανεξάρτητα από το αν επρόκειτο για σελίδες που έρχονταν ή έφευγαν από τον εκάστοτε κόμβο. Η άτακτη αυτή ενημέρωση των πινάκων σελίδων δεν προκαλούσε πρόβλημα στο αρχικό πρωτόκολλο, αφού δεν υπήρχε ανάγκη επικοινωνίας των κόμβων, εφόσον οι μεταναστεύουσες σελίδες υπήρχαν ήδη στη λανθάνουσα μνήμη. Με το νέο πρωτόκολλο, που απαιτεί και τη μετανάστευση σελίδων που δεν υπάρχουν ολόκληρες στη λανθάνουσα μνήμη, οι οποίες πρέπει άρα να μεταφερθούν μεταξύ των σταθμών, προκύπτουν τα προβλήματα συγχρονισμού που περιγράψαμε. Είναι λογικό να αναμένει κανείς, ότι ο διαχωρισμός της διαδικασίας της ενημέρωσης των πινάκων σελίδων και της μεταφοράς των σελίδων σε δύο επιμέρους διαδικασίες, ανάλογα με το αν πρόκειται για σελίδες που φεύγουν ή έρχονται

στον κάθε σταθμό, θα λύσει τέτοια προβλήματα συγχρονισμού. Με το σκεπτικό αυτό υλοποιήθηκαν δύο ξεχωριστές συναρτήσεις μετανάστευσης, οι οποίες χρησιμοποιούν τις πληροφορίες που αποθήκευσε η `migpage()` στους πίνακες για να ολοκληρώσουν τη διαδικασία της μετανάστευσης.

Πρώτη εκτελείται η συνάρτηση `void migpagesarriving()`, η οποία φροντίζει για τη μετανάστευση των σελίδων που πρόκειται να έλθουν στον εκάστοτε σταθμό. Τις πληροφορίες για αυτές τις σελίδες τις διαβάζει από τον πίνακα `pagesArriving[][]`. Η συνάρτηση `void migpagesarriving()` ελέγχει αν ο σταθμός έχει έγκυρο αντίγραφο της εκάστοτε μεταναστεύουσας σελίδας στη λανθάνουσα μνήμη του, διαφορετικά τη ζητάει από την προηγούμενη έδρα της, καλώντας την `getpage()`. Επίσης δίνει μια θέση για τη νέα σελίδα στον πίνακα σελίδων που εδράζονται στο σταθμό, αποδεσμεύει τη θέση της στον πίνακα σελίδων που φυλάσσονται στη λανθάνουσα μνήμη και ανανεώνει το πεδίο σταθμού-έδρας για τη σελίδα αυτή στον καθολικό πίνακα σελίδων. [7]

Στη συνέχεια εκτελείται η συνάρτηση `void migpagesleaving()`, η οποία αναλαμβάνει τη μετανάστευση των σελίδων που πρόκειται να φύγουν από τον εκάστοτε σταθμό. Τις πληροφορίες για τις σελίδες αυτές τις διαβάζει από τον πίνακα `pagesLeaving[][]`. Η συνάρτηση `void migpagesleaving()` αποδεσμεύει τη θέση της εκάστοτε μεταναστεύουσας σελίδας από τον πίνακα σελίδων που εδράζονται στο σταθμό, κρατά τη σελίδα στη λανθάνουσα μνήμη ή την ακυρώνει εάν δεν υπάρχει ελεύθερη θέση και ανανεώνει το πεδίο σταθμού-έδρας για τη σελίδα αυτή στον καθολικό πίνακα σελίδων.

Η κλήση των δύο συναρτήσεων γίνεται διαδοχικά και ανάμεσα τους μεσολαβεί ένα σημείο συγχρονισμού. Αυτό επιτυγχάνεται με την κλήση της συνάρτησης

[7] Δοκιμάστηκε και μια έκδοση που διαχώριζε σε διαφορετικές συναρτήσεις τις διαδικασίες αίτησης σελίδας και ανανέωσης των πινάκων σελίδων, η οποία όμως καθυστερούσε περισσότερο χωρίς να έχει κάποιο εμφανές πλεονέκτημα αξιοπιστίας.

void jia_wait() του JIAJIA , η οποία εξασφαλίζει αναμονή μέχρι όλοι οι κόμβοι να φθάσουν σε εκείνο το σημείο εκτέλεσης (είναι δηλαδή αντίστοιχη ενός φράγματος, αλλά χωρίς λειτουργίες σχετιζόμενες με τη συνοχή). Το επιπλέον αυτό σημείο συγχρονισμού, αν και εισάγει κάποια καθυστέρηση, εξασφαλίζει με απλό τρόπο ότι πρώτα θα ζητηθούν και θα σταλούν όλες οι σελίδες που πρέπει να μεταφερθούν και μετά θα γίνουν αλλαγές στους πίνακες σελίδων, θα αποαντιστοιχιθούν οι σελίδες ή θα γίνουν αναφορές σε αυτές.

Προκειμένου να αποφευχθούν τα προβλήματα με την εξυπηρέτηση των αιτήσεων λήψεων σελίδων που εμφανίζονταν στην προηγούμενη υλοποίηση, στη νέα λύση οι κλήσεις των συναρτήσεων void migpagesarriving() και void migpagesleaving() δε γίνονται στο σώμα κάποιου εξυπηρέτη. Εκεί γίνεται μόνο η αποθήκευση των πληροφοριών μετανάστευσης από τη migpage(). Οι κλήσεις των άλλων δύο συναρτήσεων και της void jia_wait() γίνονται εντός της συνάρτησης του JIAJIA που προκαλεί το φράγμα jia_barrier(), αλλά αφού έχει ολοκληρωθεί η εξυπηρέτηση του αντίστοιχου μηνύματος BARRGRANT από τον barrgrantserver(). Έτσι οι αιτήσεις για λήψη σελίδων μπορούν να εξυπηρετηθούν άμεσα.

Στη νέα υλοποίηση, οι αλλαγές στους πίνακες σελίδων των παλαιών και των νέων εδρών των σελίδων που μεταναστεύουν, καθώς και οι μεταφορές τέτοιων σελίδων δε γίνονται κατά την εκτέλεση κάποιου εξυπηρέτη. Αφού λοιπόν ο barrgrantserver() δεν είναι επιφορτισμένος με αυτόν το ρόλο, δεν υπάρχει και λόγος αναδιάταξης των εισερχόμενων μηνυμάτων, για να εξασφαλίζεται ο συγχρονισμός μέσω της σωστής διαδοχής της εκτέλεσης των εξυπηρετών.

Επιπλέον δεν υπάρχει η ανάγκη χρήσης μεταβλητών συγχρονισμού, αφού ο συγχρονισμός εξασφαλίζεται μέσω της κλήσης της jia_wait() ανάμεσα στην εκτέλεση των migpagesarriving() και migpagesleaving().

3.4.17 Επιβεβαίωση της σωστής λειτουργίας της μετανάστευσης σελίδων

Η ολοκληρωμένη διαδικασία της μετανάστευσης σελίδων που περιγράφηκε, παρακολουθήθηκε με λεπτομέρεια και σε κάθε της στάδιο, ώστε να επιβεβαιωθεί η σωστή λειτουργία της. Ο υπολογισμός των μεγεθών των διαφορών και των μέγιστων τροποποιητών επαληθεύθηκε από διάφορα σημεία του κώδικα, όπως αναλύθηκε. Οι πληροφορίες μετανάστευσης, δηλαδή οι διευθύνσεις μνήμης των ισχυρά τροποποιημένων από εξωτερικούς σταθμούς σελίδων και οι αναγνωριστές των σταθμών αυτών, που πρόκειται να γίνουν οι νέες έδρες των ανίστοιχων σελίδων μνήμης, επιβεβαιώθηκε ότι μεταδίδονται σωστά. Συγκεκριμένα, επιβεβαιώθηκε ότι παραμένουν οι ίδιες κατά τη φόρτωσή τους στο μήνυμα αίτησης φράγματος και στο μήνυμα παραχώρησης φράγματος και κατά την ανάγνωσή τους από αυτά. Επιπλέον επιβεβαιώθηκε ότι οι σωστές σελίδες ζητούνται και στέλνονται, όταν αυτό κρίνεται αναγκαίο. Παρακολουθήθηκε η ουρά εισερχόμενων μηνυμάτων, οι κλήσεις και οι εκτελέσεις των διαφόρων χειριστών σημάτων και εξυπηρετών μηνυμάτων, καθώς και οι αλλαγές στην κατάσταση προστασίας σελίδων. Οι τελευταίες παρακολουθήθηκαν προσθέτοντας ένα ακόμη όρισμα (`char *caller`) στη συνάρτηση που εκτελεί αυτή τη λειτουργία στο JIAJIA (`void memprotect(caddr_t addr,size_t len, int prot)`), ώστε να αναγνωρίζεται το σημείο και η συνάρτηση από την οποία κλήθηκε η `memprotect()` κάθε φορά.

Κεφάλαιο 4

Πειράματα και μετρήσεις

4.1 Μικρομετροπρογράμματα πρόκλησης μετανάστευσης σελίδων

4.1.1 Υλοποίηση μικρομετροπρογραμμάτων

Ακολουθώντας τη δομή μιας τυπικής εφαρμογής που χρησιμοποιεί κλήσεις στο JIAJIA [12], δημιουργήθηκε ένα μικρομετροπρόγραμμα (microbenchmark), το οποίο κάνει εκτεταμένες αναφορές σε απομακρυσμένες σελίδες μνήμης, ώστε να προκαλείται η διαδικασία της μετανάστευσης των σελίδων αυτών.

Προκειμένου το πρόγραμμα να χρησιμοποιήσει κλήσεις προς το JIAJIA συμπεριλαμβάνεται στον πηγαίο κώδικα το αρχείο `<jia.h>`. Η δομή του προγράμματος έχει ως εξής: Αφού αρχικοποιηθούν οι δομές του JIAJIA και ξεκινήσει η εκτέλεση των διεργασιών σε όλους τους σταθμούς (καλώντας τη `jia_init()`), κατανέμεται κοινή μνήμη κυκλικά, εξίσου σε όλους τους κόμβους (καλώντας τη `jia_alloc3()`). Συγκεκριμένα, κατανέμεται ένας πίνακας

χωρισμένος σε ίσα τμήματα, καθένα εκ των οποίων εδράζεται σε έναν από τους σταθμούς. Η μετανάστευση σελίδων μνήμης ενεργοποιείται καλώντας τη `jia_config(HMIG,ON)`. Η φύλαξη στατιστικών στοιχείων έχει ενεργοποιηθεί καλώντας τη `jia_startstat()`, ενώ χρόνοι μετρούνται καλώντας τη `jia_clock()`.

Στη συνέχεια ξεκινά η εκτέλεση της συνάρτησης - εργάτη. Πολλές διεργασίες εργάζονται σε κοινά δεδομένα. Κάθε διεργασία γράφει στο τμήμα του πίνακα που εδράζεται στο σταθμό στον οποίο τρέχει. Στη συνέχεια γράφει διαδοχικά στα τμήματα του πίνακα που εδράζονται στους άλλους σταθμούς. Με τη διαδικασία αυτή κάθε τμήμα του πίνακα αλλάζει έδρα. Ακολούθως κάθε διεργασία γράφει στο τμήμα του πίνακα που νωρίτερα ήταν ξένο, αλλά πλέον εδράζεται στο σταθμό που εκτελείται η διεργασία. Ο συγχρονισμός —ώστε κάθε στάδιο της διαδικασίας να διαχωρίζεται από τα υπόλοιπα— γίνεται με φράγματα. Τελικά το πρόγραμμα τερματίζει, αφού δηλώσει στο JIAJIA τον τερματισμό της εκτέλεσης, καλώντας τη `jia_exit()`, ώστε να απελευθερωθούν οι δομές του JIAJIA και ο κόμβος - διαχειριστής (master) να τερματίσει αφού έχουν τερματίσει όλοι οι άλλοι κόμβοι.

Υλοποιήθηκε ένα ακόμη μικρομετροπρόγραμμα πρόκλησης μετανάστευσης σελίδων, με σκοπό να γίνει φανερή η διαφορά του ήδη υπάρχοντος και του υλοποιηθέντος πρωτοκόλλου μετανάστευσης. Χρησιμοποιήθηκαν οι κλήσεις προς το JIAJIA που περιγράφηκαν και για το προηγούμενο μικρομετροπρόγραμμα. Η εργασία που εκτελεί η συνάρτηση - εργάτης όμως διαφοροποιείται. Πιο συγκεκριμένα έχουμε μία κοινή σελίδα (ή και περισσότερες), στην οποία όλοι οι σταθμοί αναφέρονται (τη γράφουν). Ένας από όλους τους σταθμούς γράφει μεγαλύτερο τμήμα της από όλους τους άλλους. Η σελίδα αυτή δε μεταναστεύει με το παλαιό πρωτόκολλο μετανάστευσης, μιας και δεν έχουμε την περίπτωση μοναδικού εγγραφέα. Αντίθετα μεταναστεύει με το νέο πρωτόκολλο μετανάστευσης, στο οποίο νέα έδρα γίνεται ο πιο ισχυρός τροποποιητής της σελίδας.

4.1.2 Εκτέλεση μικρομετροπρογραμμάτων

Εκτελώντας το πρώτο μικρομετροπρόγραμμα διαπιστώνουμε τη μετανάστευση όλων των σελίδων, είτε με το νέο, είτε με τον παλαιότερο μηχανισμό. Με την εκτέλεση του δεύτερου μικρομετροπρογράμματος μόνο ο νέος μηχανισμός οδηγεί σε μετανάστευση, αφού δεν έχουμε μοναδικό τροποποιητή στο διάστημα μεταξύ δύο διαδοχικών φραγμάτων. Επιβεβαιώνουμε έτσι ότι ο νέος μηχανισμός λειτουργεί σωστά, μεταναστεύοντας τις σελίδες που τον προκαλούμε να μεταναστεύσει. Μιας και δεν πρόκειται για πραγματικές εφαρμογές, αλλά για προγράμματα που φτιάχθηκαν ειδικά για να επιβεβαιωθεί η σωστή λειτουργία του μηχανισμού μετανάστευσης σελίδων, δεν έχει ιδιαίτερη αξία η παράθεση συγκριτικών επιδόσεων χρόνου ή μεταφερόμενης πληροφορίας.

4.2 Πλατφόρμες εκτέλεσης

Οι κυρίως μετρήσεις έγιναν σε συστάδα αποτελούμενη από 4 υπολογιστές με επεξεργαστές Intel Pentium III @ 866MHz (256KB λανθάνουσα μνήμη) και κύρια μνήμη 256MB. Το διασυνδετικό δίκτυο ήταν GigaBit Ethernet, το λειτουργικό σύστημα Linux (έκδοση πυρήνα 2.4.18) και ο μεταγλωττιστής gcc (έκδοση 3.2). Επιπλέον μετρήσεις έγιναν σε συστάδα αποτελούμενη από 2 υπολογιστές με επεξεργαστές Intel Pentium @ 133MHz και κύρια μνήμη 64MB. Το διασυνδετικό δίκτυο σε αυτήν την περίπτωση ήταν 100MBit Ethernet, το λειτουργικό σύστημα Linux (έκδοση πυρήνα 2.4.14) και ο μεταγλωττιστής gcc (έκδοση 2.95.2). Σε όλες τις περιπτώσεις οι μεταγλωττίσεις βιβλιοθήκης και εφαρμογών έγιναν χρησιμοποιώντας την επιλογή βελτιστοποίησης `-O2`, ενώ οι μετρήσεις έγιναν χωρίς επιπλέον κίνηση στο δίκτυο και με ελάχιστο επιπλέον φόρτο στα μηχανήματα.

4.3 Εφαρμογές

Προκειμένου να αξιολογηθούν οι επιδόσεις του JIAJIA και των βελτιώσεών μας, χρησιμοποιήθηκαν και κάποιες γνωστές εφαρμογές - προγράμματα μετρήσεων (benchmarks). Συγκεκριμένα χρησιμοποιήθηκαν τα εξής μετροπρογράμματα: Από τα SPLASH και SPLASH2 [30] τα Water, LU, από τα NAS Parallel Benchmarks [21] τα EP, IS, από τη διανομή του TreadMarks Software DSM [7] τα SOR, TSP, από τη διανομή του JIAJIA [4] το PI και αναπτυγμένο από εμάς, με βάση μια εφαρμογή του JIAJIA , το MM. Τα βασικά χαρακτηριστικά των εφαρμογών αυτών έχουν ως εξής:

- Το *Water* είναι ένα πρόγραμμα εξομοίωσης των δυνάμεων μεταξύ μοριών. Η βασική δομή δεδομένων του είναι ένας μονοδιάστατος πίνακας δομών, κάθεμια εκ των οποίων εκπροσωπεί ένα μόριο. Το Water χωρίζει στατικά τον πίνακα σε ίσα συνεχή τμήματα, καθένα εκ των οποίων ανατίθεται σε κάποιον επεξεργαστή. Η πιο απαιτητική σε χρόνο και επικοινωνία εργασία λαμβάνει χώρα στη φάση υπολογισμού των διαμοριακών δυνάμεων, κατά την οποία κάθε σταθμός υπολογίζει τις διαμοριακές δυνάμεις μεταξύ καθενός από τα μόρια που του αντιστοιχούν και καθενός από τα μισά των μορίων που τα ακολουθούν στον πίνακα. Κάθε επεξεργαστής χρησιμοποιεί ένα κλειδί αμοιβαίου αποκλεισμού για να προστατεύει την ανανέωση των τιμών των δυνάμεων των μορίων που του ανήκουν. Οι επαναλήψεις των υπολογισμών διαχωρίζονται από φράγματα.

- Το *LU* παραγοντοποιεί μια πυκνή μήτρα σε γινόμενο μιας άνω και μιας κάτω τριγωνικής μήτρας, χρησιμοποιώντας τον αλγόριθμο παραγοντοποίησης κατά τμήματα (blocks). Ο αλγόριθμος παραγοντοποιεί τη μήτρα ανά βήματα. Κάθε βήμα ολοκληρώνεται σε τρεις φάσεις, οι οποίες διαχωρίζονται με φράγματα.

- Το *EP* δημιουργεί ζεύγη Gaussian τυχαίων αποκλίσεων με ένα σχήμα κατάλληλο για παράλληλο υπολογισμό και τα πινακοποιεί διαδοχικά. Ο μόνος συγχρονισμός και επικοινωνία του EP είναι η συγκέντρωση μιας λίστας δέκα ακεραίων σε έναν κρίσιμο τομέα στο τέλος του προγράμματος.

- Το *IS* κατατάσσει μια μη διατεταγμένη ακολουθία κλειδιών, χρησιμοποιώντας ταξινόμηση με κάδους (bucket sort). Τα κλειδιά μοιράζονται στους επεξεργαστές και καθένας έχει έναν ιδιωτικό κάδο, ενώ υπάρχει και ένας κοινός για όλους κάδος. Αρχικά κάθε επεξεργαστής μετρά τα κλειδιά που βρίσκονται σε ιδιωτικό κάδο. Στη συνέχεια οι τιμές αυτές συγκεντρώνονται στον κοινό κάδο κατά τη διάρκεια ενός κρίσιμου τομέα που προστατεύεται από ένα κλειδί αμοιβαίου αποκλεισμού. Τελικά κάθε επεξεργαστής διαβάζει το άθροισμα και ταξινομεί τα κλειδιά του.

- Το *SOR* επιλύει μερικές διαφορικές εξισώσεις με τη μέθοδο της διαδοχικής υπερχαλάρωσης. Ο κόκκινος και ο μαύρος πίνακας της μεθόδου κατανείμονται σε κοινή μνήμη και χωρίζονται σε περίπου ίσες ταινίες στηλών. Κάθε επεξεργαστής υπολογίζει μια κόκκινη και μια μαύρη ταινία και συγχρονίζεται με τους άλλους επεξεργαστές με φράγματα. Επικοινωνία λαμβάνει χώρα μεταξύ των οριακών στηλών σε ένα φράγμα. Δύο φράγματα χρησιμοποιούνται για κάθε επανάληψη.

- Το *TSP* επιλύει το πρόβλημα του περιοδεύοντος πωλητή χρησιμοποιώντας έναν αλγόριθμο διακλάδωσης και περιορισμού (branch and bound). Οι βασικές κοινές δομές δεδομένων του TSP περιλαμβάνουν ένα σύνολο μερικώς αξιολογημένων διαδρομών, μια ουρά προτεραιοτήτων που περιέχει δείκτες σε διαδρομές στο σύνολο αυτό, μια στοίβα δεικτών σε αχρησιμοποίητα στοιχεία διαδρομών του συνόλου και το τρέχον συντομότερο μονοπάτι. Οι επεξεργαστές ταξινομούν τα επιμέρους μονοπάτια διαδοχικά και εκ περιτροπής, μέχρι να βρεθεί το συντομότερο μονοπάτι. Κλειδιά αμοιβαίου αποκλεισμού χρησιμοποιούνται για να διασφαλίσουν την αποκλειστική πρόσβαση στα κοινόχρηστα αντικείμενα.

- Το *PI* είναι μια απλή εφαρμογή υπολογισμού δεκαδικών ψηφίων του π. Η αποκλειστική πρόσβαση στην τρέχουσα τιμή του π εξασφαλίζεται με τη χρήση κλειδιού αμοιβαίου αποκλεισμού. Η αρχή και το τέλος της διαδικασίας υπολογισμού σηματοδοτούνται από φράγματα.

- Το *MM* υπολογίζει το εσωτερικό γινόμενο δύο πινάκων. Οι πολλαπλασιαζόμενοι πίνακες και ο πίνακας του γινομένου είναι κοινοί και μοιράζονται μεταξύ όλων των κόμβων. Η αρχικοποίηση των πινάκων που πολλαπλασιάζονται γίνεται στο σταθμό-διαχειριστή. Κάθε σταθμός αναλαμβάνει τον πολλαπλασιασμό ενός τμήματος των πινάκων. Οι ξεχωριστές φάσεις της εφαρμογής διαχωρίζονται με φράγματα, αλλά ο ίδιος ο πολλαπλασιασμός δεν απαιτεί συγχρονισμό, αφού κάθε σταθμός ασχολείται με διαφορετικό τμήμα των πινάκων.

4.4 Αξιολόγηση παράλληλου πολλαπλασιασμού στο JIAJIA

4.4.1 Περιγραφή

Έγινε μια ενδεικτική αξιολόγηση του JIAJIA ως πλατφόρμα εκτέλεσης παράλληλων προγραμμάτων και των επιδόσεών του, χωρίς να είναι ενεργοποιημένος ο μηχανισμός μετανάστευσης σελίδων μνήμης. Για το σκοπό αυτό χρησιμοποιήθηκε η εφαρμογή υπολογισμού του εσωτερικού γινομένου δύο πινάκων, με μέγεθος πινάκων $1024x1024$, σε διάφορες εκδόσεις της, ανάλογα με το σταθμό ή τους σταθμούς που φιλοξενούν αρχικά τους πίνακες που πολλαπλασιάζονται και τελικά τον πίνακα - εσωτερικό γινόμενο.

Προκειμένου να συγκριθούν οι επιδόσεις της παράλληλης εφαρμογής που αναπτύχθηκε, υλοποιήθηκε και μια σειριακή εφαρμογή πολλαπλασιασμού πινάκ-

Problem size	*Num of procs*	*Seq time(sec)*	*Par time(sec)*	*Speedup*1	*Speedup*2
1024	1	23.33	23.30	1.001	1.000
	2		12.15	1.920	1.918
	4		7.18	3.249	3.245

Πίνακας 4.1: Χρόνοι εκτέλεσης πολλαπλασιασμού πινάκων στο JIAJIA .

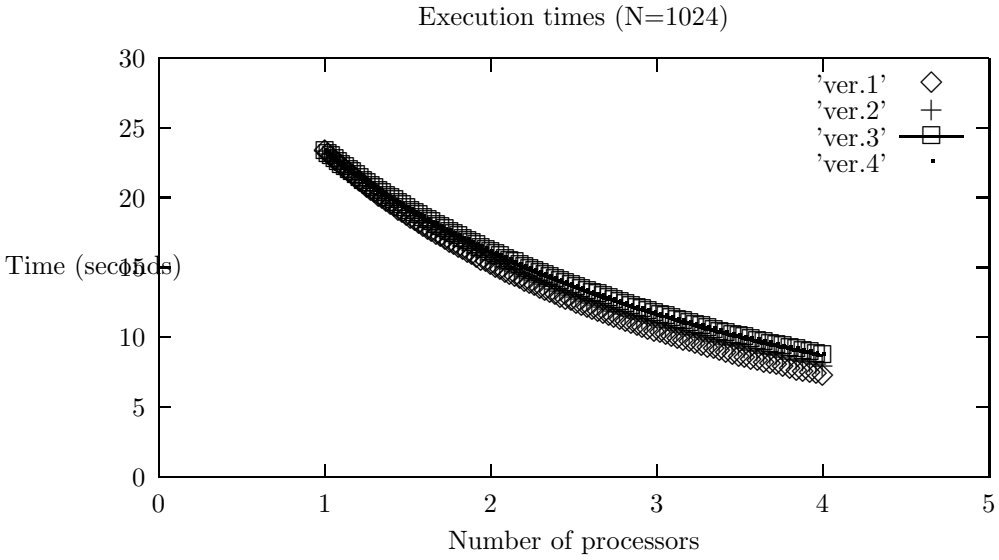
Problem size	*Num of procs*	*Total bytes received*
1024	1	0
	2	4231268
	4	19041004

Πίνακας 4.2: Συνολική παραληφθείσα πληροφορία στον πολλαπλασιασμό πινάκων στο JIAJIA .

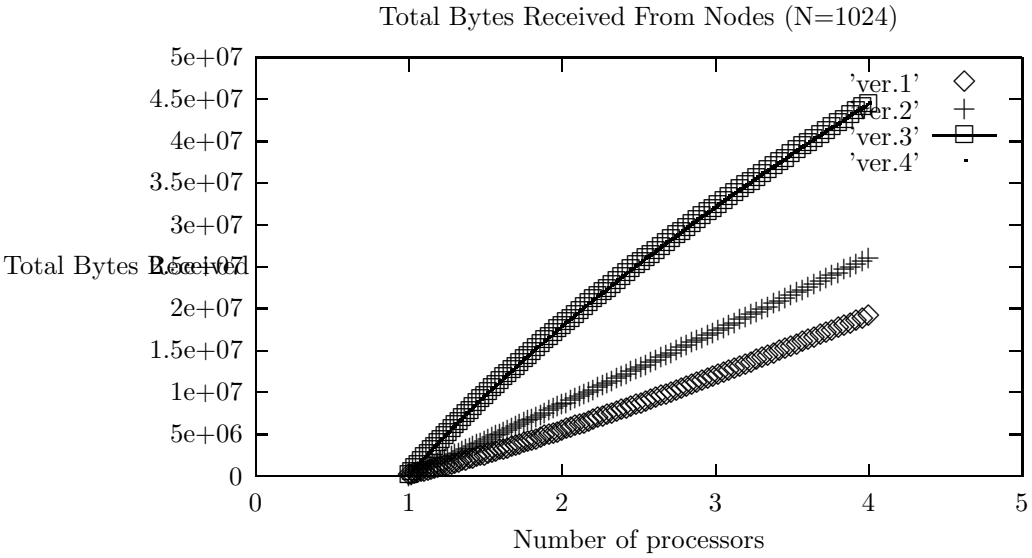
ων, που χρησιμοποιεί τον ίδιο αλγόριθμο, ο οποίος είναι καλός και σε ακολουθιακή εκτέλεση. Για τη μέτρηση χρόνου χρησιμοποιήθηκε η συνάρτηση gettimeofday που προσφέρει ακρίβεια μικροδευτερολέπτου.

4.4.2 Μετρήσεις

Ακολουθούν οι μετρήσεις με τη μορφή πινάκων και χαρακτηριστικά διαγράμματα. Η χρονοβελτιώση (speedup) σε κάθε περίπτωση υπολογίσθηκε ως ο λόγος του σειριακού χρόνου εκτέλεσης προς τον παράλληλο, όπου ο σειριακός χρόνος εκτέλεσης ήταν είτε ο χρόνος εκτέλεσης τους ακολουθιακού προγράμματος (Speedup1), είτε ο χρόνος εκτέλεσης του παράλληλου προγράμματος (με τις κλήσεις στη βιβλιοθήκη) σε ένα μόνο κόμβο (Speedup2) (τιμές που δε διαφέρουν άλλωστε πολύ).



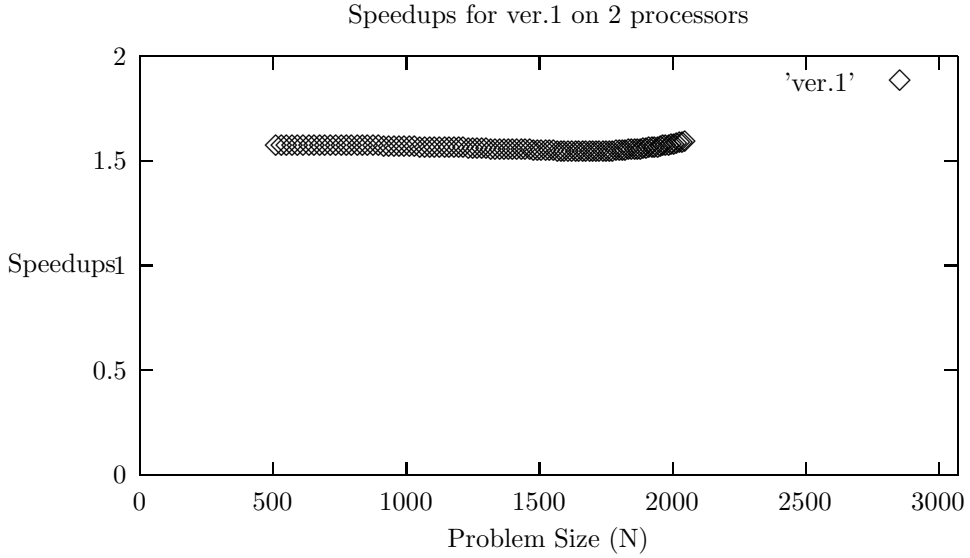
Σχήμα 4.1: Διάγραμμα χρόνων εκτέλεσης πολλαπλασιασμού πινάκων στο JIAJIA .



Σχήμα 4.2: Διάγραμμα συνολικής παραληφθείσας πληροφορίας στον πολλαπλασιασμό πινάκων στο JIAJIA .

Num of procs	*Problem size*	*Seq time(sec)*	*Par time(v.1)(sec)*	*Speedup*
2	512	15.59	9.95	1.567
	1024	146.59	93.36	1.570
	1536	496.58	319.90	1.552
	1800	760.66	506.92	1.501
	2048	1265.89	796.96	1.588

Πίνακας 4.3: Χρονοβελτιώσεις σε 2 σταθμούς στον πολλαπλασιασμό πινάκων στο JIAJIA .



Σχήμα 4.3: Διάγραμμα χρονοβελτιώσεων σε 2 σταθμούς στον πολλαπλασιασμό πινάκων στο JIAJIA .

4.4.3 Συμπεράσματα

Οι χρόνοι παράλληλης εκτέλεσης είναι αισθητά βελτιωμένοι σε σύγκριση με τους αντίστοιχους σειριακούς. Η χρονοβελτίωση που επιτυγχάνεται για 2 κόμβους είναι κοντά στην ιδανική, ενώ και για 4 κόμβους έχουμε χρονοβελτίωση αυξημένη κατά 1 ακόμη μονάδα, γεγονός που μας επιτρέπει να εξάγουμε θετικά συμπεράσματα για την κλιμακοσιμότητα του συστήματος στη συγκεκριμένη εφαρμογή. Η σημασία του διασυνδετικού δικτύου είναι προφανής, αφού χρησιμοποιώντας το JIAJIA σε περιβάλλον με αργό διασυνδετικό δίκτυο, όπως εκείνο που διασυνδέει τους 2 σταθμούς, δεν μπορούν να επιτευχθούν πολύ καλές χρονοβελτιώσεις. Όπως είναι αναμενόμενο, οι εκδόσεις του παράλληλου προγράμματος με το λιγότερο επιπλέον κόστος για μετακινήσεις πετυχαίνουν και καλύτερους χρόνους εκτέλεσης. Επίσης τα bytes που απαιτείται να μεταφερθούν αυξάνονται, καθώς αυξάνεται ο αριθμός των κόμβων που συμμετέχουν στον πολλαπλασιασμό. Σημειώνεται ότι κάθε πίνακας από doubles μεγέθους NxN αποθηκεύεται σε NxNx8bytes. Με βάση τα παραπάνω μπορούν να ερμηνευθούν τα μεγέθη της μετακινούμενης πληροφορίας.

4.5 Αξιολόγηση της ανάγκης μετανάστευσης σελίδων σε συγκεκριμένες εφαρμογές

4.5.1 Περιγραφή

Με στόχο να αξιολογηθούν τα περιθώρια βελτίωσης του υπάρχοντος πρωτοκόλλου μετανάστευσης σελίδων, μετρήθηκε (σε δύο σταθμούς) για τις διαθέσιμες εφαρμογές ο αριθμός των σελίδων που έχει νόημα να μετακινηθούν, αλλά δε μετακινούνται με το υπάρχον πρωτόκολλο. Με άλλα λόγια παρατηρήθηκε ο

Application	*Remotely modified pages*	*Migrated pages*	*Pages that could be migrated*
EP	1	0	1
IS	2	2	0
LU	130	130	0
MM	512	512	0
PI	1	1	0
SOR	2048	0	2048
TSP	95	48	47
WATER	11	9	2

Πίνακας 4.4: Ο αριθμός των σελίδων που θα μπορούσαν να μεταναστεύσουν αλλά δε μεταναστεύουν με το υπάρχων πρωτόκολλο του JIAJIA , για διάφορες εφαρμογές, σε δύο σταθμούς.

αριθμός των σελίδων που μεταβάλλονται από άλλους σταθμούς πλην της έδρας τους, αλλά δεν έχουν μοναδικό τροποποιητή στο διάστημα μεταξύ δύο φραγμάτων, ώστε να μεταναστεύσουν σε αυτόν. Ο αριθμός αυτός προκύπτει αφαιρώντας από τον αριθμό των σελίδων που τροποποιούνται από εκτός έδρας σταθμό τον αριθμό των σελίδων που μεταναστεύουν.

4.5.2 Μετρήσεις

Τα αποτελέσματα των μετρήσεων παρουσιάζονται στον πίνακα 4.4.

4.5.3 Συμπεράσματα

Παρατηρούμε ότι υπάρχουν περιθώρια βελτίωσης του υπάρχοντος πρωτοκόλλου μετανάστευσης σελίδων, κυρίως σε εφαρμογές όπως οι SOR, TSP, WATER. Ωστόσο οι απομακρυσμένες εγγραφές σελίδων δεν είναι ιδιαίτερα αυξημένες, κυρίως λόγω της φύσης των εφαρμογών, οπότε και οι σελίδες προς μετανάστευση δεν είναι γενικά πολλές. Αυτό το συμπέρασμα ενθαρρύνεται αν λάβουμε

υπ' όψιν ότι δεν αποκλείσαμε από τις μετρήσεις και σελίδες που μεταβάλλονται ελάχιστα. Βεβαίως σε περισσότερους σταθμούς οι απομακρυσμένες εγγραφές σελίδων και οι σελίδες που έχει αξία να μεταναστεύσουν είναι πιθανά περισσότερες, αν και πάλι όχι ιδιαίτερα πολλές στις συγκεκριμένες εφαρμογές.

4.6 Αξιολόγηση των μηχανισμών μετανάστευσης σελίδων σε συγκεκριμένες εφαρμογές

4.6.1 Περιγραφή

Έγιναν μετρήσεις προκειμένου να συγκριθεί η απόδοση των διαθέσιμων εφαρμογών χωρίς να χρησιμοποιείται ο μηχανισμός μετανάστευσης σελίδων, με την απόδοσή τους με χρήση του υπάρχοντος μηχανισμού μετανάστευσης σελίδων και με την απόδοσή τους με χρήση του μηχανισμού μετανάστευσης σελίδων που υλοποιήθηκε. Οι μετρήσεις αφορούσαν χρόνους, αλλά και ποσότητα πληροφορίας που μετακινείται μεταξύ των σταθμών.

4.6.2 Παράμετροι βελτιστοποίησης απόδοσης

Σημαντικές παράμετροι για την απόδοση των εφαρμογών με το νέο μηχανισμό μετανάστευσης είναι τόσο ο ορισμός κατάλληλης τιμής στο κατώφλι μετανάστευσης, όσο και η ακριβής θέση ενεργοποίησης και απενεργοποίησης του μηχανισμού. Το κατώφλι μετανάστευσης (χαρακτηριστικό μόνο του νέου πρωτοκόλλου μετανάστευσης) βοηθά στην αποφυγή του κόστους μεταναστεύσεων για σελίδες που τροποποιήθηκαν ελάχιστα από απομακρυσμένους σταθμούς, ακόμη και αν αυτοί ήταν οι μοναδικοί τροποποιητές τους. Επιπλέον συμβάλλει στην

αποφυγή του φαινομένου της παλινδρόμησης σελίδων μεταξύ των ίδιων σταθμών ως έδρες, αφού έμμεσα επιτρέπει να λαμβάνονται υπ' όψιν και οι πιθανές τοπικές τροποποιήσεις από την έως τότε έδρα κάθε σελίδας. Στην παρούσα φάση, το κατώφλι υπολογίζεται εμπειρικά από τον προγραμματιστή εφαρμογών και τίθεται καλώντας τη συνάρτηση `jia_config(HMIGthreshold, int value)`. Η τιμή του κατωφλίου εξαρτάται κύρια από το διασυνδετικό δίκτυο, τη μνήμη και την επεξεργαστική ισχύ των σταθμών και μπορεί να αφεθεί και στο προεπιλεγμένο «0» αν θέλουμε να επιτρέπονται όλες οι προτεινόμενες μεταναστεύσεις.

Η ακριβής θέση ενεργοποίησης και απενεργοποίησης του μηχανισμού μετανάστευσης είναι επίσης σημαντικός παράγοντας για τη βέλτιστη απόδοση αυτού. Ασφαλώς ο μηχανισμός πρέπει να ενεργοποιείται μετά την αρχικοποίηση των κοινών δεδομένων, ώστε ο αρχικοποιητής να μη γίνει έδρα όλων των κοινών σελίδων. Συχνά είναι χρήσιμο να απενεργοποιείται ο μηχανισμός μετά από ένα σημείο της εκτέλεσης της εφαρμογής, ώστε να αποφεύγεται η επιβάρυνση των μεταναστεύσεων. Στο σημείο αυτό θεωρούμε ότι οι μεταναστεύσεις έχουν επιτύχει ήδη τα κοινά δεδομένα να βρίσκονται κοντά στους σταθμούς που τα χρειάζονται περισσότερο. Μπορεί κανείς άρα να ενεργοποιεί το μηχανισμό μετανάστευσης για την πρώτη επανάληψη ενός μεγάλου βρόχου και να τον απενεργοποιεί κατόπιν. Η ενεργοποίηση του μηχανισμού μετανάστευσης γίνεται καλώντας τη συνάρτηση `jia_config(HMIG,ON)`, ενώ η απενεργοποίησή του καλώντας `jia_config(HMIG,OFF)`.

4.6.3 Μετρήσεις

Τα αποτελέσματα των μετρήσεων για διάφορες εφαρμογές παρουσιάζονται στον πίνακα 4.5 για τέσσερις σταθμούς και στον πίνακα 4.6 για δύο σταθμούς.

Measurement	*Without HMIG*	*With old HMIG*	*With new HMIG*
EP			
Execution time	72.841	74.987	73.003
Total bytes received	51584	51584	52016
Total pages used	1	1	1
Total migrated pages	0	0	0
SEGV time	0.00	0.00	0.00
Synchronization time	3.12	1.99	3.42
Server time	0.01	0.02	0.01
IS			
Execution time	0.678	0.875	0.717
Total bytes received	630884	630884	603624
Total pages used	1	1	1
Total migrated pages	0	0	1
SEGV time	0.04	0.14	0.03
Synchronization time	1.75	1.90	1.95
Server time	0.07	0.29	0.07
LU			
Execution time	25.85	25.93	26.53
Total bytes received	9704556	9704556	9719316
Total pages used	1028	1028	1028
Total migrated pages	0	0	0
SEGV time	0.52	0.55	0.57
Synchronization time	9.60	9.66	12.45
Server time	0.65	0.90	0.79
SOR			
Execution time	4.89	5.43	7.11
Total bytes received	5209072	5209036	5252488
Total pages used	4096	4096	4096
Total migrated pages	0	0	0
SEGV time	0.81	0.86	0.94
Synchronization time	1.58	2.39	10.33
Server time	0.65	1.40	0.92
PI			
Execution time	0.018858	0.022209	0.030822
Total pages used	1	1	1
Total migrated pages	0	0	1

Πίνακας 4.5: Συγκριτικές μετρήσεις επιδόσεων για τα διαφορετικά πρωτόκολλα μετανάστευσης, για διάφορες εφαρμογές, σε 4 σταθμούς.

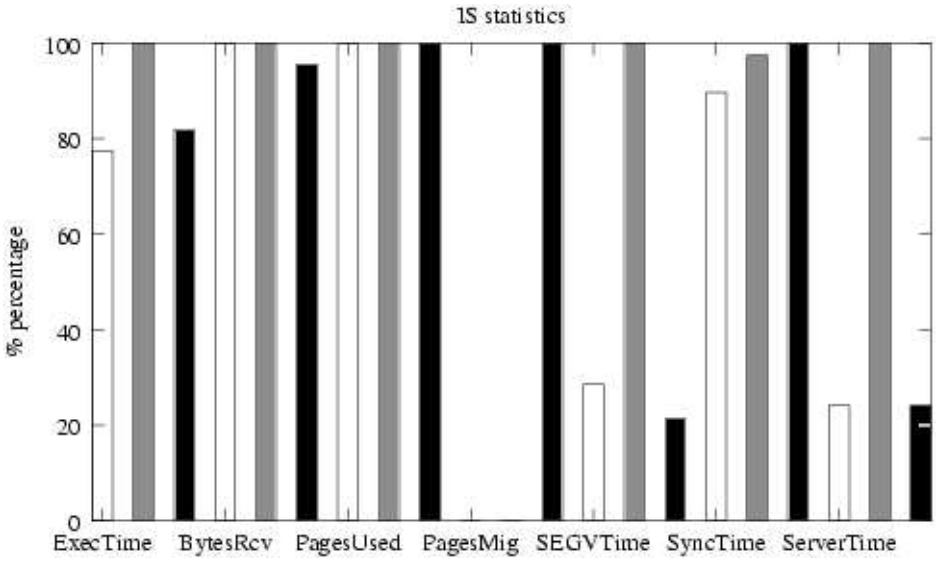
Measurement	*Without HMIG*	*With old HMIG*	*With new HMIG*
MM			
Execution time	1.14	1.19	1.20
Total bytes received	528996	1586824	1587984
Total pages used	384	384	384
Total migrated pages	0	0	128
SEGV time	0.03	0.08	0.03
Synchronization time	0.05	0.04	0.12
Server time	0.03	0.10	0.11
TSP			
Execution time	321.68	255.05	294.19
Total bytes received	10456016	8415000	11056336
Total pages used	99	99	99
Total migrated pages	0	1	3
SEGV time	159.54	122.29	140.20
Synchronization time	82.50	64.74	126.26
Server time	233.20	177.45	220.19

Πίνακας 4.6: Συγκριτικές μετρήσεις επιδόσεων για τα διαφορετικά πρωτόκολλα μετανάστευσης, για διάφορες εφαρμογές, σε 2 σταθμούς.

Οι μετρήσεις του IS ως χαρακτηριστική εφαρμογή παρουσιάζονται στο διάγραμμα 4.4. Έχοντας ποσοστά επί τοις εκατό η σύγκριση της απόδοσης των διαφόρων μηχανισμών μετανάστευσης μπορεί να γίνει εύκολα.

4.6.4 Συμπεράσματα

Παρατηρώντας τους πίνακες 4.6 και 4.5 και το διάγραμμα 4.4 εξάγονται αρκετά χρήσιμα συμπεράσματα, τόσο για το είδος των εφαρμογών, όσο και για τις συγκριτικές επιδόσεις του νέου μηχανισμού μετανάστευσης. Είναι σημαντική η παρατήρηση ότι οι εφαρμογές που ήταν διαθέσιμες δεν προσφέρονται για την ανάδειξη των πλεονεκτημάτων του μηχανισμού μετανάστευσης σελίδων. Χρησιμοποιούν λίγες σελίδες κοινής μνήμης και άρα λίγες σελίδες επίσης μεταναστεύουν. Πέρα από αυτό οι αναφορές σε απομακρυσμένες σελίδες μνήμη-



Σχήμα 4.4: Συγκριτικό διάγραμμα επιδόσεων των διαφόρων μηχανισμών μετανάστευσης για μια τυπική εφαρμογή (IS) σε 4 σταθμούς. (Λευκό: Χωρίς μετανάστευση, Γκρίζο: Μετανάστευση με το υπάρχον πρωτόκολλο, Μαύρο: Μετανάστευση με το νέο πρωτόκολλο.)

ς είναι σχετικά περιορισμένες και άρα η μεταφερόμενη πληροφορία μεταξύ των σταθμών μικρή. Συνεπώς τα πλεονεκτήματα που προσφέρει η μετανάστευση σελίδων δεν είναι εμφανή. Ακόμη οι χρόνοι εκτέλεσης είναι σχετικά μικροί, όπως και τα bytes της ανταλλασσόμενης πληροφορίας. Λόγω των μικρών μεγεθών η αξία των συμπερασμάτων είναι περιορισμένη, μιας και σε μικρή κλίμακα μεγεθών το σχετικό πειραματικό σφάλμα είναι μεγαλύτερο. Τέλος δεν έχουν όλες οι εφαρμογές μεγάλη ανάγκη συγχρονισμού και μάλιστα με φράγματα, ώστε να δίνουν συχνά τη δυνατότητα στο μηχανισμό μετανάστευσης σελίδων να ενεργοποιείται.

Παρόλο που οι διαθέσιμες εφαρμογές δεν προσφέρονται για απόλυτα συμπεράσματα, μια σύγκριση των μηχανισμών μετανάστευσης μεταξύ τους και με τη λειτουργία χωρίς μετανάστευση είναι θεμιτή και αρκετά ενθαρρυντική. Ο χρόνος εκτέλεσης με το νέο μηχανισμό μετανάστευσης είναι μικρότερος απ΄ ότι με τον προϋπάρχοντα και επομένως τα οφέλη από το νέο και πιο πολύπλοκο πρωτόκολ-

λο μετανάστευσης φαίνεται να αντισταθμίζουν την πιθανά μεγαλύτερη χρήση του επεξεργαστή για την εξυπηρέτηση του πρωτοκόλλου. Από την άλλη, εφόσον για τους λόγους που αναφέρθηκαν δεν αξιοποιούνται πολύ τα πλεονεκτήματα της μετανάστευσης σελίδων, ο χρόνος εκτέλεσης δεν είναι μικρότερος της απλής λειτουργίας χωρίς μετανάστευση σελίδων, η οποία προφανώς πλεονεκτεί στο ότι δεν επιβαρύνει τον επεξεργαστή με κάποιο πρωτόκολλο μετανάστευσης.

Όσον αφορά το χρόνο που αφιερώνεται σε εξυπηρετήσεις segmentation violation signals, τα αποτελέσματα είναι επίσης ενθαρρυντικά, μιας και η αποφυγή αναφορών σε απομακρυσμένες σελίδες που προσφέρει ο νέος μηχανισμός, όσο σπάνια και αν συμβαίνει, εξοικονομεί πολύ χρόνο. Από την άλλη βέβαια το πιο πολύπλοκο πρωτόκολλο προκαλεί μια μικρή καθυστέρηση στα σημεία συγχρονισμού (φράγματα) που είναι και σημεία λήψης των αποφάσεων μετανάστευσης, καθώς και στην εκτέλεση των διαφόρων εξυπηρετών.

Λόγω και της φόρτωσης των πληροφοριών μετανάστευσης σε ήδη υπάρχοντα μηνύματα, τα bytes που απαιτούνται για τη μετάδοση των πληροφοριών μετανάστευσης είναι λίγα και έτσι έστω και μια μικρή αξιοποίηση των σελίδων που μεταναστεύουν επιτυγχάνει τη συνολική εξοικονόμηση της μεταδιδόμενης πληροφορίας[1]. Το συμπέρασμα αυτό είναι ιδιαίτερα χρήσιμο, μιας και στις συστάδες υπολογιστών οι απομακρυσμένες προσβάσεις κοστίζουν ακριβά λόγω του αργού διασυνδετικού δικτύου.

Συνολικά, παρόλο που οι διαθέσιμες εφαρμογές δε βοηθούν πολύ στην ανάδειξη των πλεονεκτημάτων από τη χρήση του μηχανισμού μετανάστευσης σελίδων, τα αποτελέσματα είναι αρκετά ενθαρρυντικά, όπως δείχνει και το διάγραμμα 4.4. Ο νέος μηχανισμός μετανάστευσης σελίδων έχει καλύτερα αποτελέσματα από τον προϋπάρχοντα και ελαττώνει συνήθως το ποσό της

[1]Η ενσωμάτωση περισσότερων μεταφερόμενων σελίδων σε ένα μήνυμα θα βελτίωνε τη χρήση του εύρους ζώνης, αλλά είναι ουσιαστικά ανέφικτη, αφού οι αιτήσεις λήψης σελίδων θα πρέπει να εξυπηρετούνται άμεσα και όχι όταν έχει συμπληρωθεί ένα μήνυμα.

μεταδιδόμενης πληροφορίας, απαιτώντας παράλληλα λίγους κύκλους του επεξεργαστή για την επεξεργασία του πρωτοκόλλου. Το τελευταίο στοιχείο είναι ιδιαίτερα χρήσιμο για να κρατηθεί ψηλά το τόσο σημαντικό για τις επιδόσεις των κατανεμημένων κοινών μνημών λογισμικού συστάδων υπολογιστών ποιλίκο υπολογισμού προς επικοινωνία.

Κεφάλαιο 5

Συμπεράσματα

5.1 Ανακεφαλαίωση

Οι συστάδες υπολογιστών είναι μια σχετικά νέα και ελκυστική, λόγω του χαμηλού της κόστους και της ευελιξίας της, πλατφόρμα υπολογισμού υψηλών επιδόσεων. Ιδιαίτερα ενδιαφέρουσα προβάλλει η χρήση σε συστάδες υπολογιστών της αρχιτεκτονικής κατανεμημένης κοινής μνήμης, που παρέχοντας την ψευδαίσθηση της κοινής μνήμης, διευκολύνει το έργο του προγραμματιστή εφαρμογών. Οι κατανεμημένες κοινές μνήμες λογισμικού αν και πλεονεκτούν σε κόστος υλοποίησης δεν έχουν μέχρι σήμερα ιδιαίτερα συνήθως πολύ καλές επιδόσεις συγκριτικά με ανταγωνιστικές πλατφόρμες.

Προκειμένου μία κατανεμημένη κοινή μνήμη λογισμικού να μπορεί να προσαρμόζεται σε μεταβαλλόμενους υπολογιστικούς πόρους, αλλά και στις εγγενείς υπολογιστικές και επικοινωνιακές ανάγκες της εκάστοτε εφαρμογής είναι πολύ χρήσιμη η υποστήριξη δυναμικής μετανάστευσης δεδομένων ανάμεσα στους σταθμούς, η οποία βοηθά στη βελτίωση της τοπικότητας στις προσβάσεις στη μνήμη και άρα της απόδοσης του συστήματος.

Η παρούσα διπλωματική εργασία ασχολήθηκε με την ενσωμάτωση ενός μηχανισμού μετανάστευσης σελίδων στην κατανεμημένη κοινή μνήμη λογισμικού JIAJIA . Αντικαταστάθηκε ο συντηρητικός μηχανισμός μετανάστευσης σελίδων του JIAJIA από έναν νέο, κατά τον οποίο ο πιο ισχυρός τροποποιητής μιας σελίδας μνήμης έχει την ευκαιρία να γίνει η νέα έδρα της. Κατά την υλοποίηση καταβλήθηκε ιδιαίτερη προσπάθεια για να διατηρηθεί απλό το πρωτόκολλο και για να ελαχιστοποιηθούν οι απαιτήσεις σε μετάδοση πληροφορίας μεταξύ των σταθμών. Ιδιαίτερη προσπάθεια απαιτήθηκε επίσης για την αντιμετώπιση αναγκών συγχρονισμού που εμφανίζονταν μόνο με το νέο πρωτόκολλο μετανάστευσης σελίδων.

Η σωστή λειτουργία του νέου μηχανισμού μετανάστευσης σελίδων επιβεβαιώθηκε με τη λεπτομερή παρακολούθησή του σε κάθε στάδιο. Επιπλέον υλοποιήθηκαν μικρομετροπρογράμματα που προκαλούν τη διαδικασία της μετανάστευσης σελίδων, για να ελεγχθεί η σωστή συμπεριφορά του μηχανισμού. Οι μετρήσεις που έγιναν με πραγματικές εφαρμογές ήταν ενθαρρυντικές. Οι διαθέσιμες εφαρμογές δεν προσφέρονταν για την ανάδειξη των πλεονεκτημάτων του μηχανισμού μετανάστευσης σελίδων, λόγω κυρίως των περιορισμένων προσβάσεων σε απομακρυσμένες μνήμες, του μικρού ποσού μεταφερόμενης πληροφορίας και των μικρών χρόνων εκτέλεσης. Παρόλο που οι διαθέσιμες εφαρμογές δεν προσφέρονταν για απόλυτα συμπεράσματα, η σύγκριση του νέου μηχανισμού με τον παλαιότερο και με τη λειτουργία δίχως μηχανισμό μετανάστευσης σελίδων επιβεβαιώνει ότι με μικρό επεξεργαστικό κόστος για το πιο πολύπλοκο πρωτόκολλο καταφέρνουμε να μειώσουμε αρκετά το ποσό της μεταδιδόμενης πληροφορίας. Η μείωση της επικοινωνιακής επιβάρυνσης, μαζί με την ικανότητα για προσαρμογή στις μεταβαλλόμενες ανάγκες της εκάστοτε εφαρμογής είναι και ο βασικός στόχος του νέου μηχανισμού μετανάστευσης σελίδων και το κύριο πλεονέκτημα από τη χρήση αυτού.

5.2 Μελλοντική εργασία

Η υλοποίηση της τεχνικής της προώθησης σελίδων μνήμης στις συστάδες υπολογιστών φάνηκε ιδιαίτερα ελπιδοφόρα για το μέλλον των επιδόσεων των τελευταίων. Η αξία της μόλις διαφαίνεται, ενώ σε μελλοντική εργασία μας προτιθέμεθα πέρα από τη βελτίωση της υλοποίησης της τεχνικής να δείξουμε και τα ξεκάθαρα πλεονεκτήματά της.

Όσον αφορά τη βελτίωση της υλοποίησης της τεχνικής, πέρα από την εφαρμογή πολυπλοκότερων πολιτικών μετανάστευσης προτιθέμεθα να ενσωματώσουμε σαφή ανίχνευση και αποφυγή παλινδρομούντων μεταξύ των ίδιων σταθμών σελίδων. Χρήσιμη θα ήταν επίσης η ενσωμάτωση της δυνατότητας της ανίχνευσης και μέτρησης τοπικών προσβάσεων μνήμης μέσω πιθανά της παρακολούθησης μετρητών επίδοσης του επεξεργαστή. Ακόμη ένα μικρομετροπρόγραμμα αυτόματης εύρεσης του εκάστοτε βέλτιστου κατωφλίου μετανάστευσης και η δυνατότητα αυτόματης ενσωμάτωσης από μεταγλωττιστή των κλήσεων μετανάστευσης στα βέλτιστα σημεία του κώδικα των προγραμμάτων που χρησιμοποιούν τη βιβλιοθήκη θα διευκόλυναν το έργο του προγραμματιστή εφαρμογών. Τέλος η δυνατότητα για βέλτιστη προσαρμογή του μηχανισμού μετανάστευσης σελίδων στο πρότυπο πρόσβασης της μνήμης που έχει κάθε εφαρμογή θεωρούμε ότι θα βελτίωνε κατά πολύ τις επιδόσεις της βιβλιοθήκης.

Όσον αφορά την ανάδειξη των προτερημάτων της τεχνικής, μελλοντικός στόχος μας αποτελεί η δοκιμή της και με άλλες πραγματικές εφαρμογές, οι οποίες θα έχουν μεγαλύτερη ανάγκη των πλεονεκτημάτων που προσφέρει. Σκόπιμες κρίνονται επίσης οι δοκιμές με μεγαλύτερα μεγέθη ακόμη και στις υπάρχουσες εφαρμογές, ώστε να χρησιμοποιούνται και να μεταναστεύουν περισσότερες σελίδες μνήμης.

Βιβλιογραφία

[1] Mark Baker. *Cluster Computing White Paper*. University of Portsmouth, 2.0η έκδοση, Δεκέμβριος 2000.

[2] Donald Becker και Phil Merkey. The beowulf project. http://www.beowulf.org, 2001.

[3] Bryan Buck και Pete Keleher. Locality and performance of page- and object-based DSMs. Στο *Proc. of the First Merged Symp. IPPS/SPDP 1998)*, σελίδες 687–693, 1998.

[4] Chinese Academy of SciencesCenter of High Performance Computing, Institute of Computing Technology. Distributed shared memory. http://www.ict.ac.cn/chpc/dsm, 1999.

[5] University of California at Berkeley Computer Science Division. The berkeley now project. http://now.cs.berkeley.edu, 1995.

[6] Alan L. Cox, Sandhya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony και Willy Zwaenepoel. Software versus hardware shared-memory implementation: A case study. Στο *Proc. of the 21th Annual Int'l Symp. on Computer Architecture (ISCA'94)*, σελίδες 106–117, 1994.

[7] The TreadMarks DSM. The treadmarks distributed shared memory (DSM) system. http://www.cs.rice.edu/ willy/TreadMarks/overview.html, 2002.

[8] Sandhya Dwarkadas, Pete Keleher, Alan L. Cox και Willy Zwaenepoel. Evaluation of release consistent software distributed shared memory on emerging network technology. Στο *Proc. of the 20th Annual Int'l Symp. on Computer Architecture (ISCA'93)*, σελίδες 144–155, 1993.

[9] The GGF Forum. Global grid forum. http://www.gridforum.org, 2002.

[10] The MPI Forum. Message passing interface forum. http://www.mpi-forum.org, 1998.

[11] John L. Hennessy και David A. Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, California, 2η έκδοση, 1996.

[12] Weiwu Hu. *JIAJIA User's Manual*. Center of High Performance Computing, Institute of Computing Technology, Chinese Academy of Sciences, 2.2η έκδοση, 1999.

[13] Weiwu Hu, Weisong Shi και Zhimin Tang. The JIAJIA software DSM system. Τεχνική Αναφορά υπ. αρίθμ., Center of High Performance Computing, Institut of Computing Technology, Chinese Academy of Sciences, 1998.

[14] Weiwu Hu, Weisong Shi και Zhimin Tang. Home migration in home-based software DSMs. Στο *Proc. of the 1st Workshop on Software Distributed Shared Memory (WSDSM'99)*, 1999.

[15] Daniel S. Katz και Jeremy Kepner. Embedded/real-time systems. Sto *Cluster Computing White Paper* [1].

[16] Pete Keleher. Distributed shared memory home pages. http://www.cs.umd.edu/ keleher/dsm.html, 2002.

[17] Pete Keleher, Alan L. Cox, Sandhya Dwarkadas και Willy Zwaenepoel. An evaluation of software-based release consistent protocols. *Journal of Parallel and Distributed Computing*, 29(2):126–141, 1995.

[18] Pete Keleher, Alan L. Cox και Willy Zwaenepoel. Lazy release consistency for software distributed shared memory. Στο *Proc. of the 19th Annual Int'l Symp. on Computer Architecture (ISCA'92)*, σελίδες 13–21, 1992.

[19] Peter J. Keleher. Symmetry and performance in consistency protocols. Στο *International Conference on Supercomputing*, σελίδες 43–50, 1999.

[20] Donald G. Morris και David K. Lowenthal. Accurate data redistribution cost estimation in software distributed shared memory systems. *ACM SIGPLAN Notices*, 36(7):62–71, 2001.

[21] NAS. The NAS parallel benchmarks. http://www.nas.nasa.gov/Software/NPB/, 1997.

[22] M. C. Ng και W. F. Wong. Adaptive schemes for home-based DSM systems. Στο *Proc. of the 1st Workshop on Software Distributed Shared Memory (WSDSM'99)*, 1999.

[23] Dimitrios S. Nikolopoulos, Theodore S. Papatheodorou, Constantine D. Polychronopoulos, Jesús Labarta και Eduard Ayguadé. A case for user-level dynamic page migration. Στο *International Conference on Supercomputing*, σελίδες 119–130, 2000.

[24] Dimitrios S. Nikolopoulos, Theodore S. Papatheodorou, Constantine D. Polychronopoulos, Jesus Labarta και Eduard Ayguade. User-level dynamic page migration for multiprogrammed shared-memory multiprocessors. Στο *International Conference on Parallel Processing*, σελίδες 95–104, 2000.

[25] A.S. Poulakidas και A.K. Singh. Online replication of shared variables. Στο *17th International Conference on Distributed Computing Systems*, σελίδα 500, 1997.

[26] Ira Pramanick. High availability. Sto *Cluster Computing White Paper* [1].

[27] Umit Rencuzogullari και Sandhya Dwardadas. Dynamic adaptation to available resources for parallel computing in an autonomous network of workstations. *ACM SIGPLAN Notices*, 36(7):72–81, 2001.

[28] Weisong Shi. *Improving the Performance of Software DSM Systems*. Διδακτορική Διατριβή , Institute of Computing Technology, Chinese Academy of Sciences, 1999.

[29] Weisong Shi, Weiwu Hu, Zhimin Tang και M. Rasit Eskicioglu. Dynamic task migration in home-based software DSM systems. Τεχνική Αναφορά υπ. αρίθμ., Institut of Computing Technology, Chinese Academy of Sciences, 1999.

[30] SPLASH. Stanford parallel applications for shared memory. http://www-flash.stanford.edu/apps/SPLASH/, 2001.

[31] Thomas Sterling. An introduction to PC clusters for high performance computing. Sto *Cluster Computing White Paper* [1].

[32] Θ.Σ. Παπαθεοδώρου. *Εισαγωγή στα Παράλληλα Συστήματα*. Πανεπιστήμιο Πατρών, Πάτρα, 2000.

[33] Γ.Κ. Παπακωνσταντίνου, Θ.Α. Θεοχάρης και Π.Δ. Τσανάκας. *Συστήματα Παράλληλης Επεξεργασίας*. Εκδόσεις Συμμετρία, Αθήνα, 1994.

[34] The IEEE TFCC. IEEE task force on cluster computing. http://www.ieeetfcc.org, 2002.

[35] Kritchalach Thitikamol και Pete Keleher. Thread migration and communication minimization in DSM systems. *The Proceedings of the IEEE*, Μάρτιος 1999.

Ευρετήριο