ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Αποδοτική Μεταφορά Δεδομένων σε Κατανεμημένα Μέσα Αποθήκευσης Υψηλών Επιδόσεων αξιοποιώντας Δικτυακές Τεχνολογίες ATA-Over-Ethernet και Myrinet.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ιωάννη Δ. Δούδαλη

Επιβλέπων: Νεκτάριος Κοζύρης
Επίκουρος Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2006

Εθνικο Μετσοβιο Πολυτεχνειο
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Αποδοτική Μεταφορά Δεδομένων σε Κατανεμημένα Μέσα Αποθήκευσης Υψηλών Επιδόσεων αξιοποιώντας Δικτυακές Τεχνολογίες ATA-Over-Ethernet και Myrinet.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ιωάννη Δ. Δούδαλη

Επιβλέπων:Νεκτάριος Κοζύρης
Επίκουρος Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την $20^{η}$ Ιουλίου 2006

.....	.....	.....
Ν.Κοζύρης	Κ.Ζ.Πεκμεστζή	Τ.Σελλής
Επ.Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2006

.....
Ιωάννης Δ. Δούδαλης

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Τα μοντέρνα αποθηκευτικά συστήματα έχουν ξεφύγει από τα στενά όρια του τοπικού δίσκου, και σήμερα πλέον μιλάμε για κατανεμημένα αποθηκευτικά συστήματα (Storage Area Networks), που μας επιτρέπουν, μέσω ειδικών δικτύων διασύνδεσης, την επικοινωνία με απομακρυσμένες συσκευές αποθήκευσης. Ένα τέτοιο σύστημα είναι και το vRAID που αναπτύσσεται από το εργαστήριο Υπολογιστικών Συστημάτων.

Σκοπός αυτής της διπλωματικής εργασίας είναι να μελετήσει την συμπεριφορά και την επίδοση υφισταμένων πρωτοκόλλων επικοινωνίας που συνδυάζονται με I/O στον σκληρό δίσκο. Έναρξη της μελέτης μας αποτελεί το ATA over Ethernet (AoE), πρωτόκολλο που προσπαθεί να αυξήσει τις επιδόσεις παρακάμπτοντας τα ενδιάμεσα επίπεδα του TCP/IP. Στην εργασία υλοποιείται ένας server για το AoE, καθώς και διεπαφές επικοινωνίας με απομακρυσμένους δίσκους για το vRAID βασισμένες στο Ethernet και στο Myrinet.

Τα αποτελέσματα έδειξαν ότι η καλύτερη λύση είναι το Myrinet που προσφέρει τόσο το μέγιστο throughput όσο και την ελάχιστη καθυστέρηση. Αντίθετα οι λύσεις πάνω από το Ethernet περιορίζονται από το MTU του πρωτοκόλλου, και δεν καταφέρνουν να αξιοποιήσουν τις μέγιστες δυνατότητες του δικτύου.

Λέξεις Κλειδιά

Συστήματα Αποθήκευσης, Συσκευές Αποθήκευσης, Network Attached Storate, Storage Area Network, RAID, vRAID, Ata over Ethernet, Myrinet, GM, myL3.

Abstract

Modern storage systems have gone beyond local hard disks. In order to offer high performance and capacity implementations, Storage Area Networks have been created. Such a system is vRAID that is under development by Computing Systems Laboratory.

The subject of this thesis is the evaluation of the performance and characteristics of existing communication protocols that interact with the hard disk. Such a protocol is ATA over Ethernet (AoE), which tries to improve performance by bypassing the TCP/IP network stack. For my thesis I developed an AoE server and communication modules for remote disks in vRAID. Implementations over Ethernet and Myrinet are studied.

The results of the performance tests proved that the best solution is the one implemented over Myrinet, which provides high throughput and low latency results. Implementations over Ethernet suffered heavily by the MTU parameter of the protocol.

Keywords

Storage Systems, Storage Devices, Network Attached Storage, Storage Area Network, RAID, vRAID, Ata over Ethernet, Myrinet, GM, myL3.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κεφάλαιο 1

Αποθηκευτικά Συστήματα

1.1 Εισαγωγή

Ένα από τα βασικά υποσυστήματα ενός υπολογιστικού συστήματος είναι αυτό της αποθήκευσης. Κύριος εκπρόσωπός τους είναι ο σκληρός δίσκος, μια συσκευή που ήρθε να προσφέρει ασφαλή, στατική αποθήκευση δεδομένων, εξού και ο όρος "σκληρός", έναντι τις παλιάς "μαλακής" δισκέτας. Με τα χρόνια όμως μετατράπηκε στον φτωχό συγγενή του επεξεργαστή, παραμένοντας αργός, χωρίς σημαντικές αλλαγές στην αρχιτεκτονική του, και αποτελώντας στην ουσία την σημαντικότερη αιτία καθυστέρησης στην επίδοση ενός υπολογιστικού συστήματος. Το μόνο που καταφέραμε είναι να αυξάνουμε συνέχεια το μέγεθός του. Έτσι είναι πλέον αδύνατον να τον αποχωριστούμε, αφού οι υπόλοιπες διαθέσιμες λύσεις αποθήκευσης υπολείπονται είτε σε μέγεθος, είτε σε ταχύτητα, είτε ακόμα και σε αξιοπιστία.

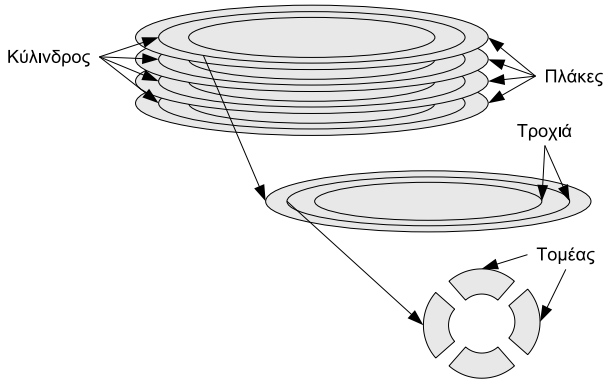
Ένα ακόμα δυνατό σημείο του σκληρού δίσκου, είναι το κόστος του, που παραμένει χαμηλό. Ήταν μονόδρομος λοιπόν η προσπάθεια δημιουργίας διατάξεων δίσκων με σκοπό την αύξηση της συνολικής προσφερόμενης ταχύτητας. Έτσι προέκυψαν τα συστήματα RAID, που συνδύασαν αποτελεσματικά φτηνούς δίσκους, για την δημιουργία γρηγορότερων και μεγαλύτερων αποθηκευτικών χώρων, που προσφέρουν επίσης και ασφάλεια στις σπάνιες, αλλά καταστροφικές, αστοχίες υλικού.

Πλέον οι ανάγκες αποθηκευτικού χώρου έχουν ξεπεράσει αυτές που μπορεί να προσφέρει ένας μόνο υπολογιστής, και έχουμε περάσει στην εποχή των κατανεμημένων αποθηκευτικών συστημάτων, που στοχεύουν να προσφέρουν Terabytes αποθηκευτικού χώρου, για να αντιμετωπίσουν τις ανάγκες των μοντέρνων εφαρμογών. Στα πλαίσια αυτής της διπλωματικής εργασίας θα εξετάσουμε τις επιδόσεις υπαρχόντων πρωτοκόλλων επικοινωνίας, όταν αυτά χρησιμοποιούνται για την διασύνδεση απομακρυσμένων δίσκων με το αποθηκευτικό

σύστημα vRAID που αναπτύσσει το εργαστήριο

Το παρόν κεφάλαιο θα αφιερωθεί για την γενική περιγραφή των αποθηκευτικών συστημάτων και του vRAID. Στο δεύτερο κεφάλαιο θα αναφερθούμε στον τρόπο λειτουργίας των πρωτοκόλλων διασύνδεσης που θα μας απασχολήσουν. Στο τρίτο κεφάλαιο θα ασχοληθούμε με τον σχεδιασμό των υποσυστημάτων επικοινωνίας. Στα κεφάλαια τέσσερα και πέντε περιγράφονται οι λεπτομέρειες της υλοποίησης του vRAID και των υποσυστημάτων μας. Τέλος στο κεφάλαιο έξι και επτά θα κάνουμε σύγκριση των διαφόρων υλοποιήσεων, και θα δούμε ποια μπορεί να είναι τα μελλοντικά πεδία έρευνας.

1.2 Μέσα Αποθήκευσης - Μαγνητικοί δίσκοι



Σχήμα 1.1: Εσωτερική δομή ενός σκληρού δίσκου.

Τα διαθέσιμα μέσα αποθήκευσης στα υπολογιστικά συστήματα είναι πολυάριθμα, εξυπηρετώντας το καθένα διαφορετικό σκοπό. Έτσι έχουμε μαγνητικά μέσα, όπως οι σκληροί δίσκοι και οι δισκέτες, οπτικά μέσα όπως οι δίσκοι DVD/CD, κάρτες μνήμης, μαγνητικές ταινίες κ.α.

Βασικό εξάρτημα όμως κάθε υπολογιστή είναι ο σκληρός δίσκος, καθώς προσφέρει την ασφαλή και σταθερή αποθήκευση των δεδομένων. Οι σκληροί δίσκοι επιτελούν δύο σκοπούς στα υπολογιστικά συστήματα.

- Μακροπρόθεσμη , διατηρήσιμη μνήμη αποθήκευσης αρχείων , ακόμη και όταν δε λειτουργεί ο υπολογιστής.

- Ένα επίπεδο στην ιεραρχία μνήμης κάτω από την κύρια μνήμη που χρησιμοποιείται ως υποστηρικτικός χώρος, για την εικονική μνήμη του λειτουργικού συστήματος, κατά την διάρκεια εκτέλεσης των προγραμμάτων.

Η εσωτερική δομή ενός δίσκου φαίνεται στο σχήμα 1.2. Ένας μαγνητικός δίσκος αποτελείται από μια σειρά από πλάκες, οι οποίες περιστρέφονται γύρω από έναν άξονα, με ταχύτητα 3600 έως 15000 στροφές ανά λεπτό (RPM). Αυτές οι πλάκες είναι μεταλλικοί ή γυάλινοι δίσκοι και είναι καλυμμένοι με μαγνητικό υλικό εγγραφής και στις δυο πλευρές τους, οπότε 10 πλάκες έχουν 20 επιφάνειες εγγραφής. Οι διάμετροι των δίσκων, το 2001, διαφέρουν κατά έναν παράγοντα του 4, από 1 έως 3.5 ίντσες, με το 95% των πωλήσεων είναι δίσκοι διαμέτρου 2.5 ή 3.5 ιντσών. Παραδοσιακά, οι μικρότεροι δίσκοι σε συνδυασμό με την χρήση καταλλήλων πρωτοκόλλων μπορούν να προσφέρουν υψηλές επιδόσεις, ενώ οι μεγάλοι δίσκοι προσφέρουν την καλύτερη απόδοση τιμής ανά MByte.

Η επιφάνεια του κάθε δίσκου διαιρείται σε ομόκεντρους κύκλους, που ονομάζονται τροχιές (tracks). Τυπικά υπάρχουν από 5000 μέχρι 30000 τροχιές σε κάθε μια από τις επιφάνειες. Κάθε τροχιά με τη σειρά της χωρίζεται σε τομείς (sectors), οι οποίοι περιέχουν την πληροφορία. Μια τροχιά μπορεί να έχει από 100 έως 500 τομείς. Ο τομέας αποτελεί τη μικρότερη μονάδα που μπορεί να εγγραφεί ή να διαβαστεί. Τα περισσότερα συστήματα ορίζουν το μέγεθος του τυπικά στα 512 bytes δεδομένων.

Προκειμένου να διαβάσουμε και να γράψουμε πληροφορίες μέσα σε έναν τομέα, ένας κινούμενος βραχίονας, που περιέχει μια κεφαλή ανάγνωσης/εγγραφής, βρίσκεται πάνω από κάθε επιφάνεια. Αντί να αναπαραστήσουμε κάθε εγγραφή από bit ανεξάρτητα, ομάδες από bits εγγράφονται χρησιμοποιώντας κωδικοποίηση "περιορισμού τρέχοντος μήκους". Οι κώδικες αυτοί διασφαλίζουν ότι υπάρχει ένας ελάχιστος και ένας μέγιστος αριθμός από bits μιας ομάδας, που ο αναγνώστης θα πρέπει να αποκωδικοποιήσει πριν να δει σήματα συγχρονισμού, πράγμα το οποίο κάνει δυνατή την υψηλότερη πυκνότητα εγγραφής και οδηγεί σε μείωση του ρυθμού σφαλμάτων. Οι βραχίονες όλων των επιφανειών συνδέονται μεταξύ τους και κινούνται συζευγμένα, έτσι ώστε όλοι οι βραχίονες να βρίσκονται πάνω στην ίδια τροχιά όλων των επιφανειών. Ο όρος κύλινδρος, χρησιμοποιείται για να αναφερόμαστε σε όλες τις τροχιές κάτω από τους βραχίονες όλων των επιφανειών σε μια δεδομένη χρονική στιγμή.

1.3 Διεπαφές Επικοινωνίας Συστημάτων Αποθήκευσης

Για τον σκοπό της διασύνδεσης των αποθηκευτικών συσκευών, τα πρωτόκολλα που επικράτησαν, και συνεχώς βελτιώνονται είναι το ATA και το SCSI. Το πρώτο απευθύνεται στη μεγάλη μάζα των αγοραστών, παρέχοντας χαμηλή τιμή, αλλά και χαμηλότερες επιδόσεις, σε σχέση με το SCSI. Το SCSI εξαιτί-

ας και του μεγάλου του κόστους χρησιμοποιείται σε λύσεις όπου οι επιδόσεις είναι σημαντικές.

Πρότυπο	ATA	SCSI	SATA	SAS
Ταχύτητα MB/sec	133	320	300	300 full-dublex
Μέγιστο μήκος καλωδίου	46cm	12m	2m	6m
Μέγιστος αριθμός συσκευών	2	16	1	16.256

Πίνακας 1.1: Χαρακτηριστικά των πρωτοκόλλων επικοινωνίας.

Εξελίξεις των δύο παραπάνω προτύπων είναι τα SATA και SAS. Το κοινό χαρακτηριστικό τους είναι ότι και τα δύο πρόκειται για σειριακά πρωτόκολλα, γιατί μόνο με αυτό τον τρόπο καταφέρουν να επιτύχουν υψηλότερους ρυθμούς διαμεταγωγής. Μάλιστα το SAS είναι συμβατό με το SATA, δίνοντας έτσι την δυνατότητα σε SATA συσκευές να συνδεθούν με τον SAS ελεγκτή, χωρίς όμως να επιτραπεί να συμβεί το αντίθετο.

Επίσης υπάρχει και μια δικτυακή επέκταση του SCSI, το iSCSI, που στην ουσία μεταφέρει πακέτα SCSI πάνω από το TCP/IP, και εξάγει τους τοπικούς δίσκους σε απομακρυσμένα συστήματα, υπό την μορφή block devices. Οι επικριτές του επισημαίνουν το επιπλέον φορτίο που εισάγει η υλοποίησή του πάνω από το υπάρχον TCP/IP. Όμως το γεγονός, ότι στην ουσία χρησιμοποιεί το δημοφιλές Ethernet, και με τις υψηλές ταχύτητες που μπορούν να επιτύχουν οι τελευταίες του εκδόσεις (10 Gigabit Ethernet), αποτελεί σημαντικό αντίπαλο για λύσεις όπως το Fibre Channel, που κυριαρχούν στην αγορά των SAN.

1.4 ATA

Το πρωτόκολλο ATA είναι ένα από τα παλαιότερα πρότυπα που υπάρχουν στον χώρο τον υπολογιστών, και επιτρέπει την επικοινωνία μεταξύ των διαφόρων μέσων αποθήκευσης (σκληρών δίσκων , cd-roms) με το υπόλοιπο υπολογιστική σύστημα. Η πλήρης περιγραφή του πρωτοκόλλου μπορεί να βρεθεί στο παρακάτω κείμενο [1]. Το πρότυπο είναι πάρα πολύ εκτενές και περιγράφει από τον τρόπο καλωδίωσης μέχρι μια μεγάλη πλειάδα εντολών. Για το σκοπό του παρόντος κειμένου, θα αναφερθούν και θα περιγραφούν, μόνο ένα μικρό υποσύνολο των βασικών εντολών, με σκοπό να είναι ευκολότερη στη συνέχεια η κατανόηση του πρωτοκόλλου ATA over Ethernet.

Καταχωρητής	Πρόσφατη εγγραφή	Προηγούμενη εγγραφή
Features	Δεσμευμένος	Δεσμευμένος
Sector Count	Sector count (7:0)	Sector count (15:8)
LBA Low	LBA (7:0)	LBA (31:24)
LBA Mid	LBA (15:8)	LBA (39:32)
LBA High	LBA (23:16)	LBA (47:40)
Device register	Bits 7 και 5 είναι παρωχημένα το LBA bit παίρνει την τιμή 1 . το DEV bit θα υποδεικνύει την επιλεγμένη συσκευή, τα bit (3:0) είναι δεσμευμένα	Δεσμευμένα

Πίνακας 1.2: Διαμόρφωση καταχωρητών για διευθύνσεις των 48 bit

Καταχωρητής	Πρόσφατη εγγραφή	Προηγούμενη εγγραφή
Features	na	na
Sector Count	Sector count (7:0)	na
LBA Low	LBA (7:0)	na
LBA Mid	LBA (15:8)	na
LBA High	LBA (23:16)	na
Device register	LBA (27:24)	na

Πίνακας 1.3: Διαμόρφωση καταχωρητών για διευθύνσεις των 28 bit

Ένα από τα καινούργια χαρακτηριστικά που εισήγαγε το νέο πρότυπο ATA 6 σε σχέση με την προηγούμενη έκδοση, είναι η επέκταση της ικανότητας διευθυνσιοδότησης των sectors ενός δίσκου κατά 20 bits. Έτσι ενώ το προηγούμενο πρότυπο υποστήριζε το LBA (η φυσική διεύθυνση του πρώτου sector προς ανάγνωση) να έχει μέγεθος 28 bit, τώρα έχει επεκταθεί στα 48. Ακόμη ο αριθμός του καταχωρητή sector count, ο μέγιστος δηλαδή αριθμός sectors που μπορεί να μεταφερθεί, έχει γίνει πλέον 16 bit. Για να είναι δυνατή η υποστήριξη των παραπάνω επεκτάσεων όσες συσκευές υποστηρίζουν το LBA 48 θα πρέπει να αυξήσουν τους διαθέσιμους καταχωρητές. Το πρότυπο υποδεικνύει οι υπάρχοντες καταχωρητές να υλοποιούνται ως ουρές FIFO των 2 bytes.

Παρακάτω θα περιγραφούν τρεις μόνο εντολές από το σύνολο που υποστηρίζει το ATA/ATAPI, γιατί είναι οι βασικές εντολές που θα χρειαστούμε για να κατανοήσουμε στην συνέχεια την λειτουργία του AoE. Οι εντόλες READ και READ-ext έχουν όμοια μορφή με αυτή των αντοστοίχων WRITE.

1.4.1 Εντολή IDENTIFY DEVICE

Σκοπός αυτής της εντολής είναι, η συσκευή να κάνει γνωστά στο σύστημα τα χαρακτηριστικά της. Ο κωδικός της εντολής είναι ο 0xEC. Η μορφή του πακέτου εισόδου είναι αυτή του σχήματος 1.2α, με το bit DEV να υποδηλώνει την επιλεγμένη συσκευή. Σε επιτυχή ολοκλήρωση της εντολής το σύστημα θα λάβει μια απάντηση μήκους 512 bytes που θα περιγράφει κατά το πρότυπο όλα τα χαρακτηριστικά της συσκευής και τις δυνατότητες που υποστηρίζει. Τα bit BSY, DF ,DRQ ,ERR θα έχουν τιμή μηδέν, το DEV τιμή ανάλογη της επιλεγμένης συσκευής, ενώ το bit DRDY θα έχει την τιμή 1 υποδηλώνοντας ότι η εντολή έχει ολοκληρωθεί.

1.4.2 Εντολές WRITE WRITE-EXT

Register	7	6	5	4	3	2	1	0
Features	na							
Sector Count	Sector count							
LBA Low	LBA (7:0)							
LBA Mid	LBA (15:8)							
LBA High	LBA (23:16)							
Device	obs	LBA	obs	DEV	LBA (27:24)			
Command	30h							

(α) Περιεχόμενο καταχωρητών της εντολής write

Register	7	6	5	4	3	2	1	0
Error	na							
Sector Count	na							
LBA Low	na							
LBA Mid	na							
LBA High	na							
Device	obs	na	obs	DEV	na	na	na	na
Status	BSY	DRDY	DF	na	DRQ	na	na	ERR

(β) Περιεχόμενο καταχωρητών μετά από επιτυχή εκτέλεση write

Register	7	6	5	4	3	2	1	0
Error	na	WP	MC	IDNF	MCR	ABRT	NM	na
Sector Count	na							
LBA Low	LBA (7:0)							
LBA Mid	LBA (15:8)							
LBA High	LBA (23:16)							
Device	obs	na	obs	DEV	LBA (27:24)			
Status	BSY	DRDY	DF	na	DRQ	na	na	ERR

(γ) Περιεχόμενο καταχωρητών μετά από αποτυχημένη εκτέλεση write

Σχήμα 1.2: Η μορφή των διαφόρων πακέτων της εντολής write μορφής LBA24

Register		7	6	5	4	3	2	1	0
Features	Current	Reserved							
	Previous	Reserved							
Sector Count	Current	Sector count (7:0)							
	Previous	Sector count (15:8)							
LBA Low	Current	LBA (7:0)							
	Previous	LBA (31:24)							
LBA Mid	Current	LBA (15:8)							
	Previous	LBA (39:32)							
LBA High	Current	LBA (23:16)							
	Previous	LBA (47:40)							
Device		obs	LBA	obs	DEV	Reserved			
Command		34h							
NOTE – The value indicated as Current is the value most recently written to the register. The value indicated as Previous is the value that was in the register before the most recent write to the register.									

(α) Περιεχόμενο καταχωρητών της εντολής write-ext

Register		7	6	5	4	3	2	1	0
Error		na							
Sector Count	HOB = 0	Reserved							
	HOB = 1	Reserved							
LBA Low	HOB = 0	Reserved							
	HOB = 1	Reserved							
LBA Mid	HOB = 0	Reserved							
	HOB = 1	Reserved							
LBA High	HOB = 0	Reserved							
	HOB = 1	Reserved							
Device		obs	na	obs	DEV	Reserved			
Status		BSY	DRDY	DF	na	DRQ	na	na	ERR
NOTE – HOB = 0 indicates the value read by the host when the HOB bit of the Device Control register is cleared to zero. HOB = 1 Indicates the value read by the host when the HOB bit of the Device Control register is set to one.									

(β) Περιεχόμενο καταχωρητών μετά από επιτυχημένη εκτέλεση write-ext

Register		7	6	5	4	3	2	1	0
Error		na	WP	MC	IDNF	MCR	ABRT	NM	obs
Sector Count	HOB = 0	Reserved							
	HOB = 1	Reserved							
LBA Low	HOB = 0	LBA (7:0)							
	HOB = 1	LBA (31:24)							
LBA Mid	HOB = 0	LBA (15:8)							
	HOB = 1	LBA (39:32)							
LBA High	HOB = 0	LBA (23:16)							
	HOB = 1	LBA (47:40)							
Device		obs	na	obs	DEV	Reserved			
Status		BSY	DRDY	DF	na	DRQ	na	na	ERR
NOTE – HOB = 0 indicates the value read by the host when the HOB bit of the Device Control register is cleared to zero. HOB = 1 Indicates the value read by the host when the HOB bit of the Device Control register is set to one.									

(γ) Περιεχόμενο καταχωρητών μετά από αποτυχημένη εκτέλεση write-ext

Σχήμα 1.3: Τα πακέτα της εντολής write-ext μορφής LBA48

Η μορφή των μηνυμάτων των δύο αυτών εντολών φαίνονται στα σχήματα 1.2 και 1.3.

1.4.2.1 Εντολή write

Η εντολή αυτή έχει κωδικό 30h και είναι υποχρεωτική για όσες συσκευές υλοποιούν το σύνολο εντολών PACKET.

1.4.2.1.1 Πακέτο εισόδου

- Sector Count: Ο αριθμός των sector που πρέπει να μεταφερθούν. Αν έχει την τιμή 00h τότε θα μεταφερθούν 256 sectors.
- LBA Low: Το εύρος των LBA bit (7:0).
- LBA Mid: Το εύρος των LBA bit (15:8).
- LBA High: Το εύρος των LBA bit (23:16).
- Device: Το bit LBA έχει την τιμή 1 για να καθορίσει ότι η διεύθυνση είναι LBA. Το DEV bit θα καθορίσει την επιλεγμένη συσκευή και τα bit (3:0) αντιστοιχούν στα LBA bit (27:24).

1.4.2.1.2 Απάντηση

- Device Register: Δηλώνει ποια συσκευή είναι επιλεγμένη.
- Status register:
  - BSY: Θα έχει την τιμή μηδέν για να υποδηλώνει την ολοκλήρωση της εντολής
  - DRDY: Θα έχει την τιμή μηδέν
  - DF (Device Fault): Θα έχει την τιμή μηδέν.
  - DRQ: Θα έχει την τιμή μηδέν
  - ERR: Θα έχει την τιμή μηδέν

1.4.2.1.3 Μήνυμα Σφάλματος

- Error register
  - WP: Θα έχει την τιμή ένα, αν πρόκειται για αφαιρούμενο μέσο που έχει προστασία εγγραφής.

  - MC: θα έχει την τιμή ένα, αν το αφαιρούμενο μέσο έχει αλλάξει κατά την εκτέλεση της τελευταίας εντολής. Η συσκευή θα αρχικοποιήσει την κατάσταση αλλαγής του μέσου.
  - IDNF: Θα έχει την τιμή ένα, αν μια προσπελάσιμη διεύθυνση δε μπόρεσε να βρεθεί. Θα έχει επίσης την τιμή ένα αν μια μη προσβάσιμη εντολή προσπάθησε να προσπελαστεί, και το ABRT bit δεν έχει την τιμή ένα.
  - MCR: Θα πάρει την τιμή ένα, αν για κάποιο αφαιρούμενο μέσο έχει γίνει αίτηση αλλαγής. Το bit λαμβάνει την τιμή μηδέν μόνο μετά την εκτέλεση της εντολής GET MEDIA STATUS, ή κάποιας άλλης εντολής πρόσβασης.
  - ABRT: Θα πάρει την τιμή ένα, αν η εντολή δεν υποστηρίζεται ή αν ένα σφάλμα, συμπεριλαμβανομένου το ICRC, έχει συμβεί κατά την εκτέλεση μιας Ultra DMA μεταφοράς δεδομένων. Θα πάρει επίσης την τιμή ένα αν ζητηθεί διεύθυνση πέραν του εύρους που είναι προσβάσιμες από τον χρήστη και αν το bit IDNF δεν έχει τη τιμή ένα.
  - NM: Θα πάρει την τιμή ένα, αν δεν έχει εισαχθεί το αφαιρούμενο μέσο.

- LBA Low, LBA Mid , LBA High, Device: Θα έχουν την διεύθυνση του πρώτου σφάλματος.
  - DEV: Θα περιγράφει την επιλεγμένη συσκευή.
- Status register:
  - BSY: Θα έχει την τιμή μηδέν, υποδηλώνοντας την ολοκλήρωση της εκτέλεσης της εντολής.
  - DRDY: Θα έχει την τιμή ένα.
  - DF: Θα έχει την τιμή ένα, αν έχει συμβεί κάποιο σφάλμα.
  - DRQ: Θα έχει την τιμή μηδέν.
  - ERR: Θα έχει την τιμή ένα αν κάποιο bit του καταχωρητή Error έχει τιμή ένα.

1.4.2.2 Εντολή Write EXT

Η εντολή αυτή έχει κωδικό 34h και είναι υποχρεωτική για όσες συσκευές υλοποιούν την δυνατότητα διευθυνσιοδότησης κάνοντας χρήση 48-bit.

1.4.2.2.1 Πακέτο εισόδου

- Sector Count Current: Ο αριθμός των sector που πρέπει να μεταφερθούν bit (7:0).
- Sector Count Previous Ο αριθμός των sector που πρέπει να μεταφερθούν bit (15:8).
- LBA Low Current: Το εύρος των LBA bit (7:0).
- LBA Low Previous: Το εύρος των LBA bit (31:24).
- LBA Mid Current: Το εύρος των LBA bit (15:8).
- LBA Mid Previous: Το εύρος των LBA bit (39:32).
- LBA High Current: Το εύρος των LBA bit (23:16).
- LBA High Previous: Το εύρος των LBA bit (47:40).
- Device: Το bit LBA έχει την τιμή 1 για να καθορίσει ότι η διεύθυνση είναι LBA. Το DEV bit θα καθορίσει την επιλεγμένη συσκευή.

1.4.2.2.2 Απάντηση

- Device Register: Δηλώνει ποια συσκευή είναι επιλεγμένη.
- Status register:
  - BSY: Θα έχει την τιμή μηδέν, για να υποδηλώνει την ολοκλήρωση της εντολής
  - DRDY: Θα έχει την τιμή μηδέν.
  - DF (Device Fault): Θα έχει την τιμή μηδέν.
  - DRQ: Θα έχει την τιμή μηδέν.
  - ERR: Θα έχει την τιμή μηδέν.

1.4.2.2.3 Μήνυμα Σφάλματος

- Error register
  - WP: Θα έχει την τιμή ένα, αν πρόκειται για αφαιρούμενο μέσο που έχει προστασία εγγραφής

  - MC: Θα έχει την τιμή ένα, αν το αφαιρούμενο μέσο έχει αλλάξει κατά την εκτέλεση της τελευταίας εντολής. Η συσκευή θα αρχικοποιήσει την κατάσταση αλλαγής του μέσου.
  - IDNF: Θα έχει την τιμή ένα, αν μια προσπελάσιμη διεύθυνση δε μπόρεσε να βρεθεί. Θα έχει επίσης την τιμή ένα αν μια μη προσβάσιμη εντολή προσπάθησε να προσπελαστεί, και το ABRT bit δεν έχει την τιμή ένα.
  - MCR: Θα πάρει την τιμή ένα, αν για κάποιο αφαιρούμενο μέσο έχει γίνει αίτηση αλλαγής. Το bit λαμβάνει την τιμή μηδέν μόνο μετά την εκτέλεση της εντολής GET MEDIA STATUS, ή κάποιας άλλης εντολής πρόσβασης.
  - ABRT: Θα πάρει την τιμή ένα, αν η εντολή δεν υποστηρίζεται ή αν ένα σφάλμα, συμπεριλαμβανομένου το ICRC, έχει συμβεί κατά την εκτέλεση μιας Ultra DMA μεταφοράς δεδομένων. Θα πάρει επίσης την τιμή ένα αν ζητηθεί διεύθυνση πέραν του εύρους που είναι προσβάσιμες από τον χρήστη και αν το bit IDNF δεν έχει τη τιμή ένα.
  - NM: Θα πάρει την τιμή ένα, αν δεν έχει εισαχθεί το αφαιρούμενο μέσο

- LBA Low, LBA Mid , LBA High, Device: Θα έχουν την διεύθυνση του πρώτου σφάλματος, και η αντιστοιχία των bit της διεύθυνσης με τους καταχωρητές είναι όμοια με αυτή του μηνύματος της αίτησης.

  - DEV: Θα περιγράφει την επιλεγμένη συσκευή.

- Status register

  - BSY: Θα έχει την τιμή μηδέν υποδηλώνοντας την ολοκλήρωση της. εκτέλεσης της εντολής.
  - DRDY: Θα έχει την τιμή ένα.
  - DF: Είναι όμοια με αυτή που δίνεται στην περίπτωση μιας εντολής Write. Έχει την τιμή ένα αν έχει συμβεί κάποιο σφάλμα.
  - DRQ: Θα έχει την τιμή μηδέν.
  - ERR: Θα έχει την τιμή ένα αν κάποιο bit του καταχωρητή Error έχει τιμή ένα.

1.5 RAID

Ήδη από τα τέλη της δεκαετίας του '80 έχει διαπιστωθεί ότι το χάσμα μεταξύ της ταχύτητας του επεξεργαστή και της μνήμης, σε σχέση με αυτή των υπόλοιπων αποθηκευτικών συστημάτων άρχισε να αυξάνεται απειλητικά, με τα τελευταία να αποτελούν στην ουσία την στενωπό του συστήματος. Παράλληλα γίνονται προσπάθειες για την ανάπτυξη μεγάλων αποθηκευτικών συστημάτων, που θα μπορούν να προσφέρουν καλύτερες επιδόσεις, σε σχέση με της υπάρχουσες λύσεις, αλλά να είναι και συμφέρουσες οικονομικά. Με το άρθρο του ο Patterson et al [21] παρουσιάζει μια συστηματική περιγραφή των έως τότε προσπαθειών, για τη δημιουργία συστοιχιών σκληρών δίσκων, προτείνοντας επιπλέον τη χρήση κοινών δίσκων για τη δημιουργία Redundant Arrays of Inexpensive Disks (RAID) [7][17] .

Τα χαρακτηριστικά αυτών των συστημάτων είναι τα εξής:

- **Ασφάλεια** Η βασική προσέγγιση του συστήματος είναι να οργανώνει τους δίσκους σε ομάδες ασφαλείας, με κάθε ομάδα να έχει επιπλέον δίσκους που θα περιέχουν πλεονάζουσα πληροφορία. Σκοπός είναι όταν κάποιος δίσκος τεθεί εκτός λειτουργίας, να είμαστε σε θέση να μπορέσουμε να τον αντικαταστήσουμε μέσα σε κάποιο περιορισμένο χρονικό διάστημα, και η "χαμένη" πληροφορία, να μπορέσει να ξαναδημιουργηθεί στον νέο δίσκο με την χρήση της πλεονάζουσας (redundant) πληροφορίας που υπάρχει στο σύστημα. Ο μέσος χρόνος που χρειάζεται για την επιδιόρθωση του συστήματος ορίζεται ως MTTR (mean time to repair). Το MTTR μπορεί να μειωθεί, αν το σύστημα διαθέτει επιπλέον δίσκους σε αναμονή, που θα αναλάβουν αυτόματα να αντικαταστήσουν τους χαλασμένους δίσκους.

- **Επιπρόσθετο Κόστος Ασφάλειας** Είναι το ποσοστό των επιπλέον δίσκων που λειτουργούν ως εφεδρικοί ως προς αυτούς που χρησιμοποιούνται για την αποθήκευση των δεδομένων. Το ποσοστό αυτό κυμαίνεται από 100% μέχρι 4%.

- **Ποσοστό Διαθέσιμης Χωρητικότητας** Είναι το ποσοστό της διαθέσιμης χωρητικότητας προς αποθήκευση σε σχέση με το συνολικό μέγεθος των δίσκων που χρησιμοποιούνται στην συστοιχία. Η τιμή του κυμαίνεται μεταξύ 50% έως 96%.

- **Επίδοση** Επειδή αυτά τα συστήματα απευθύνονται σε λύσεις υπερυπολογιστών ή βάσεων δεδομένων (transaction-processing systems), χρησιμοποιούνται διαφορετικές μέθοδοι για να αξιολογηθεί η επίδοση των αποθηκευτικών συστημάτων σε κάθε περίπτωση. Για τους υπερυπολογιστές μετριέται ο αριθμός των εγγραφών και αναγνώσεων για "μεγάλα"

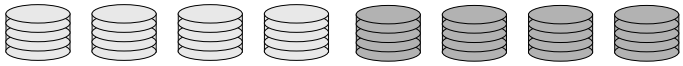
blocks δεδομένων, όπου με τον όρο "μεγάλα" υποδηλώνουμε, ότι από κάθε δίσκο δεδομένων μιας ομάδας δίσκων θα έχουμε την μεταφορά τουλάχιστον ενός sector. Κατά την μεταφορά μεγάλων blocks κάθε δίσκος σε μια ομάδα λειτουργεί ως αυτόνομη μονάδα, και έτσι η λειτουργία της εγγραφής ή της ανάγνωσης γίνονται παράλληλα σε όλους τους δίσκους.

Για συστήματα βάσεων δεδομένων μια καλύτερη μέθοδος μέτρησης της επίδοσης είναι ο αριθμός των εγγραφών και των αναγνώσεων ανά δευτερόλεπτο. Επειδή αυτά τα συστήματα εμφανίζουν ακολουθίες της μορφής ανάγνωση-μετατροπή-εγγραφή περιλαμβάνεται και αυτή η παράμετρος μέτρησης στο σύστημα. Ιδεατά κατά την διάρκεια μικρών μεταφορών κάθε δίσκος σε για ομάδα μπορεί να λειτουργεί αυτόνομα, γράφοντας ή διαβάζοντας αυτόνομα. Εν κατά κλείδει τα συστήματα υπερυπολογιστών χρειάζονται μεγάλους ρυθμούς μεταγωγής δεδομένων, ενώ οι βάσεις δεδομένων απαιτούν μεγάλους ρυθμούς λειτουργιών I/O.

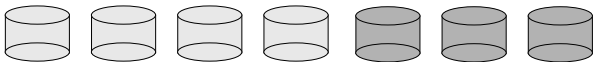
- **Πραγματική απόδοση ανά δίσκο**. Το κόστος των δίσκων είναι μια σημαντική παράμετρος ενός συστήματος βάσεων δεδομένων. Έτσι η επίδοση I/O ανά δίσκο, αποτελεί τον λόγο κόστους προς απόδοση σε ένα σύστημα.

1.5.1 Διατάξεις RAID

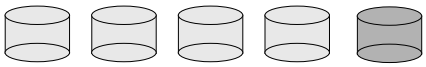
(α) Χωρίς Πλεονασμό RAID 0



(β) Κατοπτρισμός RAID 1



(γ) Memory-Style ECC RAID 2

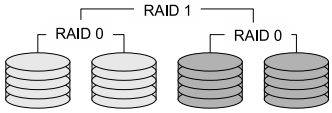


(δ) Ισοτιμία Διαφυλλωμένων Bit Parity RAID 3

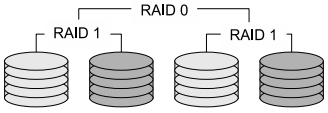
(ε) Ισοτιμία Διαφυλλωμένου Μπλοκ RAID 4

(στ) Ισοτιμία Κατανεμημένου Διαφυλλωμένου Μπλοκ RAID 5

(ζ) Πλεονασμός P+Q RAID 6



(η) Ιεραρχία RAID 0+1



(θ) Ιεραρχία RAID 1+0

Σχήμα 1.4: Στα παραπάνω σχήματα φαίνονται τα διάφορα επίπεδα του RAID. Τα πιο έντονα σκιασμένα τμήματα δείχνουν την πλεονάζουσα πληροφορία. Δίσκοι με πολλαπλά επίπεδα υποδηλώνουν διαμοιρασμό σε επίπεδο block, ενώ αυτοί χωρίς, σε επίπεδο Bit. Περισσότερες πληροφορίες για τις διάφορες μπορούν να βρεθούν στο http://en.wikipedia.org/wiki/Redundant_array_of_independent_disks

Στη συνέχεια θα περιγραφούν αναλυτικά οι διαθέσιμες υλοποιήσεις RAID.

1.5.1.1 RAID 0

Η διάταξη (σχήμα 1.4α) δε θα μπορούσε να θεωρηθεί ότι ανήκει σε συστήματα RAID, καθώς δεν διαθέτει καθόλου πλεονάζουσα πληροφορία. Η πληροφορία κατανέμεται εξίσου σε όλους του δίσκους, δημιουργώντας έναν "μεγάλο" ενιαίο δίσκο. Το πλεονέκτημα αυτής της διάταξης είναι ότι έχει το χαμηλότερο κόστος σε σχέση με την συνολική χωρητικότητά που προσφέρει, και επιπλέον παρουσίαζει την υψηλότερη επίδοση για τις εγγραφές, γιατί δεν χρειάζεται να ανανεώσει την πλεονάζουσα πληροφορία του συστήματος. Παραδόξως δεν παρουσιάζει την καλύτερη επίδοση στις αναγνώσεις. Σχήματα πλεονασμού που δημιουργούν αντίγραφα των δεδομένων, όπως το RAID 1 έχουν καλύτερες επιδόσεις, γιατί μπορούν και χρονοδρομολογούν επιλεκτικά τις αιτήσεις στους δίσκους, που παρουσιάζουν την μικρότερη καθυστέρηση αναζήτησης και περιστροφής. Σε περίπτωση βέβαια που καταστραφεί κάποιος από τους δίσκους, τότε έχει καταστραφεί ολόκληρη η συστοιχία. Η διάταξη αυτή επιλέγεται σε συστήματα υπερυπολογιστών όπου υπάρχει μεγαλύτερο ενδιαφέρον στις επιδόσεις και όχι στην ασφάλεια των δεδομένων.

1.5.1.2 RAID 1

Η κλασική λύση 1.4β του κατοπτρισμού χρησιμοποιεί τους διπλάσιους δίσκους σε σχέση με την λύση του RAID 0. Τα δεδομένα που γράφονται στον έναν δίσκο αναπαράγονται τα ίδια και στον πλεονάζοντα δίσκο, και έτσι υπάρχουν πάντα δύο αντίγραφα της ίδιας πληροφορίας. Όταν τα δεδομένα πρέπει να διαβαστούν, τότε επιλέγεται ο δίσκος με την μικρότερη καθυστέρηση. Αν κάποιος δίσκος αντιμετωπίζει πρόβλημα, τότε μπορεί να χρησιμοποιηθεί ο πλεονάζον δίσκος για την εξυπηρέτηση των αιτήσεων . Η διάταξη αυτή χρησιμοποιείται κυρίως σε συστήματα, όπως οι βάσεις δεδομένων, όπου η διαθεσιμότητα του συστήματος και ο ρυθμός απάντησης των αιτήσεων είναι πιο σημαντικός από την αποτελεσματική αξιοποίηση του διαθέσιμου αποθηκευτικού χώρου.

1.5.1.3 RAID 2

Αυτή η διάταξη πλέον δεν βρίσκει εφαρμογή σε εμπορικά συστήματα και συνήθως δεν αναφέρεται πλέον, γιατί παρουσιάζει μεγάλο κόστος υλοποίησης, αφού απαιτεί πάρα πολλούς δίσκους, και δεν έχει και τις αναμενόμενες επιδόσεις. Για την διατήρηση της ακεραιότητας του συστήματος κάνει χρήση κωδικών επιδιόρθωσης σφαλμάτων Hamming, που περιέχουν πληροφορία ισοτιμίας για διακριτά επικαλυπτόμενα υποσύνολα από αντικείμενα. Μια εκδοχή

αυτού του συστήματος είναι ότι χρειάζονται όπως στο σχήμα: 1.4γ τρεις επιπλέον δίσκοι για ένα σύστημα τεσσάρων δίσκων. Το γεγονός ότι ο αριθμός των πλεοναζόντων δίσκων πρέπει να είναι ο λογάριθμος του συνολικού αριθμού δίσκων του συστήματος, οδηγεί ώστε η αποδοτικότητα του συστήματος αυξάνει όσο αυξάνει ο αριθμός των δίσκων.

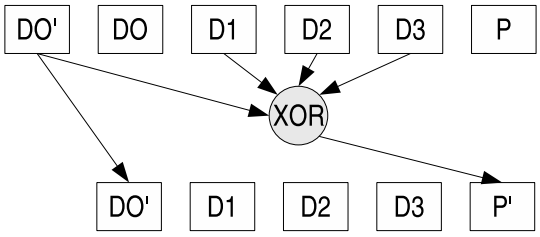
Αν καταστραφεί κάποιο δίσκος, αρκετά από τα στοιχεία ισοδυναμίας θα έχουν άκυρα δεδομένα, και το κατεστραμμένο στοιχείο είναι το κοινό κάθε υποσυνόλου. Η κατεστραμμένη πληροφορία μπορεί να ανακτηθεί διαβάζοντας τα υπόλοιπα στοιχεία σ' ένα υποσύνολο συμπεριλαμβανομένου και του στοιχείου πλεονασμού. Έτσι χρειάζονται πολλαπλοί πλεονάζοντες δίσκοι για να μπορέσουμε να εντοπίσουμε την χαμένη πληροφορία, αλλά μόνο ένα για την ανάκτησή της.

1.5.1.4 RAID 3

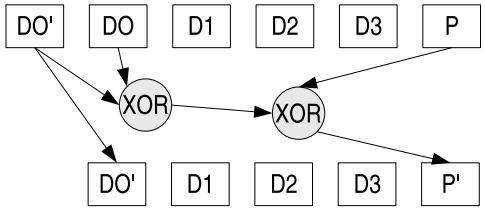
Το RAID 2 χρησιμοποιεί έναν μεγάλο αριθμό από δίσκους για να είναι σε θέση να διαπιστώσει αν κάποιος από αυτούς έχει υποστεί βλάβη. Όμως είμαστε σε θέση με την χρήση hardware να διαπιστώσουμε ποιος δίσκος έχει τεθεί εκτός λειτουργίας. Έτσι αρκεί ένας μόνον δίσκος για να αποθηκεύεται η πλεονάζουσα πληροφορία που χρειάζεται για την αποκατάσταση της βλάβης. Στην ισοτιμία διαφυλλομένων Bit, τα δεδομένα κατανέμονται κατά σειρά bit στους δίσκους δεδομένων, και προστίθεται ένας μοναδικός δίσκος ισοτιμίας, που χρησιμοποιείται για την αποκατάσταση του συστήματος, στην περίπτωση που μόνο ένας δίσκος τεθεί εκτός λειτουργίας. Κάθε αίτηση ανάγνωσης προσπελαύνει όλους τους δίσκους δεδομένων, και κάθε εγγραφή επίσης όλους τους δίσκους καθώς και τον δίσκο ισοτιμίας. Άρα μόνο μια αίτηση μπορεί να εξυπηρετηθεί την φορά. Επειδή ο δίσκος ισοτιμίας δεν περιέχει καθόλου πληροφορία, δε μπορεί να συμμετάσχει στις αναγνώσεις, παρέχοντας χαμηλότερες επιδόσεις από τις διατάξεις που κατανέμουν την πλεονάζουσα πληροφορία σε όλους τους δίσκους. Τέτοια συστήματα μπορούν να χρησιμοποιηθούν σε εφαρμογές που δεν έχουν υψηλές απαιτήσεις διαμεταγωγής και είναι πιο εύκολα υλοποιήσιμες από τις λύσεις RAID 4, 5, και 6.

1.5.1.5 RAID 4

Η ισοτιμία διαφυλλωμένου block (σχήμα 1.4ε) σε σχέση με την αντίστοιχη που γίνεται σε επίπεδο Bit, διαφέρει στο ότι πλέον η πληροφορία κατανέμεται στους δίσκους σε επίπεδο block και όχι Bit. Το μέγεθος του block αποκαλείται μονάδα διαμοιρασμού (stripping unit). Αιτήσεις ανάγνωσης μικρότερες της μονάδας διαμοιρασμού εξυπηρετούνται από την προσπέλαση ενός μόνου δίσκου. Οι αιτήσεις ανάγνωσης θα πρέπει να ανανεώσουν τα δεδομένα για τα



(α) Ανανέωση της ισοτιμίας μικρής εγγραφής RAID3 (Reconstruct-Write)



(β) Ανανέωση της ισοτιμίας μικρής εγγραφής RAID 4 ,5 (Read-Modify-Write)

Σχήμα 1.5: Εγγραφή - Ανανέωση block ισοτιμίας

οποία έχει γίνει η αίτηση αλλά είναι υποχρεωμένες επιπλέον να υπολογίσουν και το καινούργιο block ισοτιμίας. Για μεγάλες εγγραφές που μεταβάλλουν block σε όλους τους δίσκους, η καινούργια ισοτιμία μπορεί να υπολογιστεί εύκολα κάνοντας την πράξη XOR ανάμεσα στα δεδομένα για κάθε δίσκο. Για μικρές εγγραφές που ανανεώνουν έναν μόνο δίσκο, η ισοτιμία προκύπτει από τη διαφορά ανάμεσα στο καινούργιο και στο παλιό block, και εφαρμόζοντας αυτή τη διαφορά στο block ισοτιμίας. Έτσι για μικρές εγγραφές χρειάζονται τέσσερις αιτήσεις I/O στους δίσκους: μια η εγγραφή των νέων δεδομένων, δύο αναγνώσεις των παλιών τιμών του block δεδομένων και ισοτιμίας και μια εγγραφή του νέου block ισοτιμίας. Αυτή η διαδικασία αναφέρεται ως read-modify-write. Το μειονέκτημα αυτής της διάταξης είναι ότι τα δεδομένα ισοτιμίας αποθηκεύονται μόνο σε έναν δίσκο που μπορεί να γίνει η στενωπός του συστήματος. Αντί αυτής της διάταξης προτιμάται το RAID 5 που κατανέμει τα block ισοτιμίας σε όλους του δίσκους.

1.5.1.6 RAID 5

Η ισοτιμία κατανεμημένου διαφυλλωμένου block (block-interleaved distrubuted-parity) αντιμετωπίζει το πρόβλημα που παρουσιάζει η διάταξη RAID-4, όπου ο δίσκος που κρατάει τα block ισοτιμίας αποτελεί τη στενωπό του συστήματος [18]. Ένα ακόμα πλεονέκτημα είναι ότι κατανέμει τα δεδομένα σε όλους

0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

Σχήμα 1.6: Η αριστερά συμμετρική τοποθέτηση της ισοτιμίας που κατά τους Lee & Katz [18] παρουσιάζει την καλύτερη επίδοση για τις διάφορες πιθανές περιπτώσεις φορτίου.

τους δίσκους. Με αυτό το τρόπο όλοι οι δίσκοι συμμετέχουν στις αναγνώσεις, γεγονός που δεν το επιτρέπεται στα RAID 4, 3 και 2. Η διάταξη έχει την καλύτερη επίδοση σε περιπτώσεις μικρών και μεγάλων αναγνώσεων και σε μικρές εγγραφές. Στην περίπτωση των μικρών εγγραφών δεν είναι αποδοτική όπως στην περίπτωση του κατοπτρισμού, γιατί θα πρέπει να πραγματοποιηθεί η διαδικασία της ανάγνωσης-τροποποίσης-εγγραφής, που είναι και η μόνη αδυναμία της διάταξης αυτής.

Η ακριβής διαδικασία της κατανομής της ισοτιμίας στους δίσκους είναι αντικείμενο έρευνας. Οι Lee & Katz [18] προτείνουν την διάταξη του σχήματος 1.6. Το πλεονέκτημα αυτής της διάταξης είναι ότι αν πρέπει να προσπελάσεις ακολουθιακά όλες τις μονάδες διαμοιρασμού (stripping units) θα αναγκαστείς να προσπελάσεις όλους τους δίσκους πριν επανέλθεις σε κάποιον δίσκο για δεύτερη φορά.

1.5.1.7 RAID 6

Η χρήση της ισοτιμίας μας δίνει την δυνατότητα να μπορούμε να προστατεύσουμε το σύστημά μας από την αποτυχία ενός μόνο δίσκου. Σε μεγάλα όμως αποθηκευτικά συστήματα όπου η χρόνος αποκατάστασης του συστήματος είναι μεγάλος, η πιθανότητα να συμβεί κάποιο δεύτερο σφάλμα δεν είναι αμελητέα. Γι' αυτό και χρειάζεται να εφαρμόσουμε πιο δυνατά συστήματα ασφαλείας και διόρθωσης λαθών. Ένα τέτοιο παράδειγμα είναι ο αποκαλούμενος πλεοναμός P+Q, που κάνει χρήση κωδικών εντοπισμού σφαλμάτων Reed-Solomon, που μας επιτρέπουν να αντιμετωπίσουμε οσεσδήποτε αποτυχίες δίσκων προσθέτοντας στο σύστημα τους αντίστοιχους πλεονάζοντες δίσκους. Η δομή της συστοιχίας μοιάζει πάρα πολύ με αυτής του RAID 5, κατανέμοντας την πλεονάζουσα πληροφορία σε όλους τους δίσκους. Επίσης λειτουργούν με τον ίδιο τρόπο στην περίπτωση μικρών εγγραφών εφαρμόζοντας την διαδικασία

read-modify-write. Σε αυτή την περίπτωση χρειάζεται να κάνουμε εξι προσπελάσεις στον δίσκο αντί για τέσσερις που κάνουμε στην περίπτωση του RAID 5, γιατί τώρα θα πρέπει να ανανεώσουμε τις πληροφορίες P και Q.

1.5.2 Κωδικοί Αποκατάστασης Διαγραφών

Στο παρακάτω τμήμα θα αναφερθούμε στους κωδικούς που χρησιμοποιούνται στην περίπτωση που κάποιος από τους δίσκους καταστραφεί. Πρέπει να επισημάνουμε ότι οι κωδικοί που χρησιμοποιούνται δεν είναι διόρθωσης σφαλμάτων (Error Correcting Codes) αλλά διόρθωσης διαγραφών (Erasure Codes) [26], γιατί σκοπός τους δεν είναι να εντοπίσουν το πιθανό λάθος που μπορεί να έχει προέλθει κατά την μετάδοση της πληροφορίας, αλλά να αποκαταστήσουν σφάλματα που έχουν προέλθει από την αποτυχία ενός μέσου αποθήκευσης.

Οι τρόποι που υπάρχουν για να αντιμετωπιστούν τέτοια προβλήματα είναι πάρα πολλοί. Ο πιο απλός είναι με την επανάληψη των δεδομένων, μια λύση αρκετά αφελής με υψηλό κόστος υλοποίησης αλλά και με υψηλές επιδόσεις. Παρακάτω θα περιγράψουμε τις τεχνικές της απλής ισοτιμίας και της κωδικοποίησης Reed-Solomon, που χρησιμοποιούνται στην υλοποίηση των RAID 5 και 6 αντίστοιχα.

1.5.2.1 Απλή Ισοτιμία

Η απλή ισοτιμία υλοποιείται πολύ εύκολα. Αν υποθέσουμε ότι έχουμε τα bytes $d_1, d_2, \ldots, d_n$ τότε το αντίστοιχο byte ισοτιμίας θα προκύψει ως $c = d_1 \oplus d_2 \oplus \ldots \oplus d_n$, όπου με τον τελεστή $\oplus$ αναφερόμαστε στον δυαδικό τελεστή XOR. Αν αλλάξουν τα περιεχόμενα κάποιου δίσκου από d_j σε d'_j τότε το byte ισοτιμίας υπολογίζεται ως εξής: $c' = c \oplus d_j \oplus d'_j$. Έστω ότι καταστρέφεται ο δίσκος i, για να μπορέσουμε να αποκαταστήσουμε τα χαμένα δεδομένα, το μόνο που χρειάζεται να κάνουμε είναι $d_i = d_1 \oplus d_2 \oplus \ldots \oplus d_{i-1} \oplus d_{i+1} \oplus \ldots \oplus d_n \oplus c$, πράξη ιδιαίτερα απλή και γρήγορη, που απαιτεί τον ελάχιστο επιπλέον χώρο για την υλοποίησή της. Το μόνο της μειονέκτημα είναι ότι μπορεί να αντιμετωπίσει μόνο μια καταστροφή. Μειονέκτημα σημαντικό όταν τα δεδομένα που πρέπει να αποκατασταθούν είναι πολύ μεγάλα σε μέγεθος και η χρονική διάρκεια που χρειάζεται για την αποκατάστασή τους είναι τέτοια, που καθιστά την πιθανότητα να συμβεί και δεύτερη βλάβη σημαντική.

1.5.2.2 Reed-Solomon

Το πρόβλημα των περισσοτέρων της μιας αποτυχίας έρχεται καλύψουν πιο εξελιγμένοι κώδικες όπως ο Reed-Solomon (RS) [27] [25] [2] που είναι σε θέση να αντιμετωπίσει μέχρι και m κατεστραμμένους δίσκους σε μία διάταξη

n+m δίσκων, όπου n είναι οι δίσκοι δεδομένων και m οι πλεονάζοντες δίσκοι. Παρακάτω θα περιγράψουμε τον τρόπο λειτουργίας του RS γενικά και για την περίπτωση που θέλουμε να αντιμετωπίσουμε μέχρι και δύο καταστροφές, όπως αυτός χρησιμοποιείται στην υλοποίηση του RAID-6.

Ο υπολογισμός της πλεονάζουσας πληροφορίας C_i απαιτεί την ύπαρξη της συνάρτησης F_i που θα εφαρμοστεί πάνω σε όλα τα δεδομένα. Στην RS κωδικοποίηση τα δεδομένα καταμερίζονται σε *λέξεις* (*words*). Το *μέγεθος* (*size*) κάθε λέξης είναι w bits, με την τιμή του να επιλέγεται από τον προγραμματιστή. Έτσι η κάθε συσκευή περιέχει $l = (k\ bytes)(\frac{8\ bits}{byte})(\frac{1\ word}{w\ bits}) = \frac{8k}{w}$ λέξεις η κάθε μία. Όπως θα δούμε παρακάτω το μήκος της λέξης καθορίζει το μέγιστο αριθμό δίσκων που μπορεί να υποστηρίξει το σύστημα. Τυπική τιμή είναι το 8 που επιτρέπει μέχρι 256 δίσκους, αν θέλουμε μεγαλύτερο αριθμό τότε μπορούμε να επιλέξουμε μήκος λέξης 16.

Για να υπολογίζουμε την πλεονάζουσα πληροφορία c_i σε κάθε δίσκο C_i εφαρμόζουμε την συνάρτηση F_i σε κάθε λέξη:

$$c_i = F_i(d_1, d_2, \ldots, d_n)$$

Αν αλλάξει κάποια λέξη d_j στην συσκευή D_j σε d'_j τότε η πλεονάζουσα πληροφορία c_i θα πρέπει να υπολογιστεί εκ νέου με την χρήση της γενικής συνάρτησης επαναφοράς $G_{i,j}$:

$$c'_j = G_{i,j}(d_j, d'_j, c_i)$$

Αν καταστραφούν m συσκευές τότε μπορούμε να επαναφέρουμε το σύστημα ακολούθως: πρώτα για κάθε κατεστραμμένη συσκευή D_j, υλοποιούμε την συνάρτηση επαναφοράς και ανακατασκευάζουμε την κατεστραμμένη συσκευή από όσες έχουν μείνει ακέραιες. Με την ολοκλήρωση αυτής της διαδικασίας επαναφέρουμε και όσες πλεονάζουσες συσκευές είχαν καταστραφεί επαναυπολογίζοντας τα περιεχόμενά τους.

Για την απλή περίπτωση όπου m=1 επανερχόμαστε στης πράξεις που είχαμε περιγράψει στην περίπτωση της απλής ακεραιότητας.

Για την περίπτωση που έχουμε περισσότερες από μια συσκευές τότε η συνάρτηση υπολογισμού της πλεονάζουσας πληροφορίας γίνεται:

$$c_i = F_i(d_1, d_2, \ldots, d_n) = \sum_{j=1}^{n} d_j, f_{i,j}$$

Έτσι αν τα δεδομένα είναι το διάνυσμα D και η πλεονάζουσα πληροφορία το διάνυσμα C τότε η συνάρτηση υπολογισμού είναι ο πίνακας F και ισχύει η συνάρτηση $FD = C$.

Σκοπός τώρα είναι να κατασκευάσουμε τον πίνακα κατανομής πληροφορίας που θα μας επιτρέψει να διορθώσουμε τα σφάλματα που θα προκύψουν. Οι

ιδιότητες αυτού του πίνακα που χρησιμοποιεί η κωδικοποίηση RS πρέπει να είναι:

- Είναι ένας $(n+m) \times n$ πίνακας.
- Ο $n \times n$ πίνακας των n πρώτων γραμμών είναι ο πίνακας ταυτότητα I.
- Οποιοσδήποτε υποπίνακας δημιουργηθεί από την διαγραφή m γραμμών είναι αντιστρέψιμος.

Για να μπορέσουμε να δημιουργήσουμε έναν τέτοιο πίνακα χρησιμοποιούμε τον πίνακα Vandermonde που ορίζεται ως i^j:

$$\begin{bmatrix} 0^0(=1) & 0^1(=0) & 0^2(=0) & \dots & 0^{n-1}(=0) \\ 1^0 & 1^1 & 1^2 & \dots & 1^{n-1} \\ 2^0 & 2^1 & 2^2 & \dots & 2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ (n+m-1)^0 & (n+m-1)^1 & (n+m-1)^2 & \dots & (n+m-1)^{n-1} \end{bmatrix}$$

και έχει την ιδιότητα ότι κάθε υποπίνακάς του που προκύπτει από την αφαίρεση m γραμμών είναι αντιστρέψιμος. Η διαδικασία μετατροπής του στον επιθυμητό πίνακα B είναι εύκολη και περιγράφεται στο [27].

Η απεικόνιση των δεδομένων γίνεται λοιπόν:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ b_{n+1,1} & b_{n+1,2} & b_{n+1,3} & \dots & b_{n+1,n} \\ b_{n+2,1} & b_{n+2,2} & b_{n+2,3} & \dots & b_{n+2,n} \\ \vdots & \vdots & \vdots & & \vdots \\ b_{n+m,1} & b_{n+m,2} & b_{n+m,3} & \dots & b_{n+m,n} \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}}_{D} = \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}}_{E}$$

Η διαδικασία της κωδικοποίησης είναι προφανής με τους δίσκους πλεονάζουσας πληροφορίας να έχουν τιμή:

$$c_i = \sum_{j=1}^{n} b_{n+i,j} d_j$$

Αν υποθέσουμε τώρα ότι καταστρέφονται m συσκευές, τότε ο πίνακας B μετατρέπεται στον πίνακα B' που είναι ένας πίνακας $n \times n$ απ'όπου έχουν

αφαιρεθεί οι γραμμές που αντιστοιχούν σε κατεστραμμένα στοιχεία, και ο πίνακας E στον πίνακα E' που περιέχει μόνο τους δίσκους που έχουν μείνει ακέραιοι. Μένει λοιπόν να λύσουμε την εξίσωση:

$$B'D = E'$$

απ' όπου μπορούμε να ανακτήσουμε τις τιμές των δίσκων δεδομένων, και στην συνέχεια να αποκαταστήσουμε τους δίσκους πλεονάζουσας πληροφορίας που έχουν καταστραφεί.

Μια σημαντική παράμετρος του προβλήματος είναι η πραγματοποίηση των πράξεων για λύση του παραπάνω συστήματος. Επειδή έχουμε να αντιμετωπίσουμε δυαδικές πράξεις με λέξεις συγκεκριμένου μήκους, οι πράξεις όπως η διαίρεση εισάγουν σφάλματα που μπορεί να μην επιτρέπουν την επίλυση του συστήματος. Γι' αυτό το λόγο και οι πράξεις θα γίνουν με την βοήθεια της θεωρίας των πεδίων Galois όπως αυτή περιγράφεται στα [27, 25, 2].

Περιγραφή της υλοποίησης του RAID-6 στο Linux μπορεί να βρεθεί στο [2]. Μια βελτιωμένη έκδοση του κώδικα Reed-Solomon περιγράφεται στο άρθρο [28]. Περισσότερες πληροφορίες για παρόμοιους κώδικες μπορούν να βρεθούν σε βιβλία σχετικά με την Θεωρία της Πληροφορίας και σε άρθρα όπως τα παρακάτω [30, 4, 13, 14, 3, 10, 9].

1.5.3 Επίδοση Συστημάτων RAID

Το πρόβλημα της σύγκρισης της επίδοσης των διαφόρων διατάξεων RAID δεν είναι τόσο απλό όσο μπορεί να φαίνεται. Στο [7] δίνεται μια λεπτομερής ανάλυση του τρόπου σύγκρισης που μπορεί να γίνει ανάμεσα στα διάφορα είδη RAID. Ο πίνακας 1.4 δίνει την επίδοση κάποιων διατάξεων RAID στις διάφορες μορφές αιτήσεων που μπορεί να προκύψουν.

Οι παράμετροι που πρέπει να ληφθούν υπόψη είναι η αξιοπιστία , η επίδοση και το κόστος. Συμπερασματικά μπορούμε να διαπιστώσουμε πολύ εύκολα ότι οι διατάξεις RAID 0 και 1 απευθύνονται σε λύσεις όπου δίνεται η μέγιστη έμφαση στις επιδόσεις και στην ασφάλεια αντίστοιχα, θυσιάζοντας όμως την αξιοπιστία του συστήματος και το συνολικό του κόστος. Αντίθετα οι υλοποιήσεις RAID 5 και 6 επιτρέπουν τον περιορισμό του κόστους προσφέροντας αυξημένη ασφάλεια, και οι επιδόσεις τους κυμαίνονται πολύ κοντά σε αυτές των 0 και 1. Βέβαια η λύση σ' ένα πολύ μεγάλο σύστημα δεν είναι ποτέ μονομερής. Ο συνδυασμός των παραπάνω ανάλογα με τις απαιτήσεις της κάθε εφαρμογής μπορεί να βελτιστοποιήσει την προσφερόμενη λύση.

	Μικρή Ανάγνωση	Μικρή Εγγραφή	Μεγάλη Ανάγνωση	Μεγάλη Εγγραφή	Διαθέσιμος Χώρος Αποθήκευσης
RAID 0	1	1	1	1	1
RAID 1	1	1/2	1	1/2	1/2
RAID 3	$1/G$	$1/G$	$(G-1)/G$	$(G-1)/G$	$(G-1)/G$
RAID 5	1	$\max(1/G, 1/4)$	1	$(G-1)/G$	$(G-1)/G$
RAID 6	1	$\max(1/G, 1/6)$	1	$(G-2)/G$	$(G-2)/G$

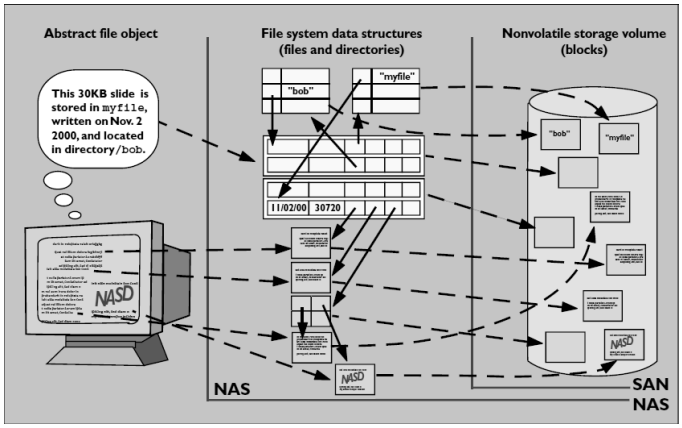
Πίνακας 1.4: Ρυθμός μετάδοσης (throughtput) ανά δολάριο συγκρινόμενα με το RAID 0 [7]

Ο παραπάνω πίνακας συγκρίνει το ρυθμό μετάδοσης των διαφόρων τύπων πλεονασμού για τέσσερις τύπους αιτήσεων I/O. Ο όρος "μικρός" αναφέρεται σε αιτήσεις που θα προσπελαστεί μόνο ένας δίσκος, ενώ ο όρος "μεγάλος" αναφέρεται σε αιτήσεις όπου θα προσπελαστούν όλοι οι δίσκοι της συστοιχίας. Ο όρος G αναφέρεται στον αριθμό των δίσκων που ανήκουν στην συστοιχία. Όσο μεγαλύτερος τόσο καλύτερα.

1.6 Αρχιτεκτονικές Δικτυακών Συστημάτων Αποθήκευσης

Παραπάνω περιγράψαμε τις διάφορες δομές που μπορούν να έχουν τα τοπικά συστήματα αποθήκευσης. Τα μειονεκτήματα τους είναι ότι ο αποθηκευτικός τους χώρος παραμένει περιορισμένος αλλά το κυριότερο ότι δεν μπορεί να είναι προσπελάσιμος από άλλους υπολογιστές. Λύση στο παραπάνω πρόβλημα δίνουν τα συστήματα δικτυακής αποθήκευσης που επιτρέπουν σε απομακρυσμένους υπολογιστές να συνδέονται με αυτά και να ανταλλάσσουν δεδομένα. Οι λύσεις που προσφέρονται, εκτείνονται σε μεγάλο εύρος με έμφαση πλέον να δίνεται στην έρευνα για κατανεμημένα αποθηκευτικά συστήματα τα οποία θα μπορούν να διαχειριστούν μεγάλα ποσά πληροφορίας.

Οι δύο αρχιτεκτονικές συστημάτων αποθήκευσης που εμφανίζονται σήμερα [12] είναι η Storage Area Networks (SAN) και η Network Attached Storage (NAS) και παρουσιάζουν τάσεις σύγκλισης, προσφέροντας ολοκληρωμένα κατανεμημένα αποθηκευτικά συστήματα. Κύρια διαφορά τους είναι ότι τα συστήματα NAS προσφέρουν μία διεπαφή επικοινωνίας, συνήθως κάποιο δικτυακό σύστημα αρχείων (NTFS, SAMBA), ενώ τα SAN επικοινωνούν σε πολύ χαμηλότερο επίπεδο (αυτό του block), προσφέροντας πολλαπλές διαπαφές διασύνδεσης. Αποτέλεσμα είναι ότι μέχρι πρότινος τα SAN να έχουν ταυτιστεί με εξειδικευμένα δίκτυα διασύνδεσης όπως το Fibre Channel, που επιτρέπουν



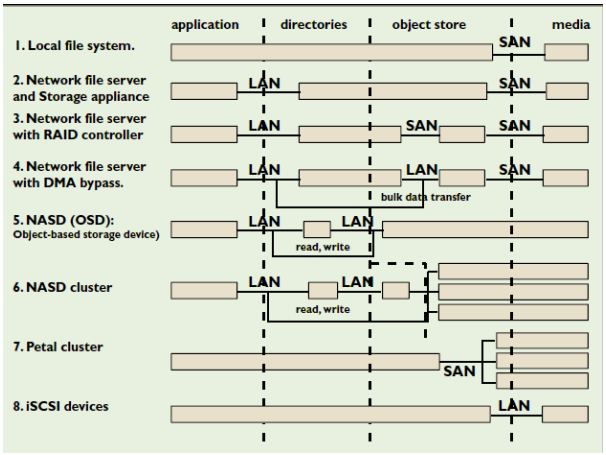
Σχήμα 1.7: Η εικόνα δείχνει την αλληλεπικάλυψη των διαθεσίμων των αρχιτεκτονικών NAS και SAN.

την γρήγορη και αποτελεσματική επικοινωνία σε επίπεδο block, όντας βελτιστοποιημένα για τέτοιες εφαρμογές, και είναι ιδιαίτερα δαπανηρά. Αντίθετα τα NAS, έχοντας προκύψει και παλιότερα, προκρίνονται ως φτηνές λύσεις, επιτρέποντας την χρήση κοινών δικτυακών λύσεων όπως το Ethernet, έχοντας παράλληλα ως κύριο μειονέκτημα το γεγονός ότι δεν είναι επεκτάσιμες.

Πλέον όμως έχουν προκύψει νέες απαιτήσεις για τις επερχόμενες λύσεις δικτυακής αποθήκευσης [12], και είναι:

- **Γεωγραφικά κατανεμημένα συστήματα** Λόγω των εφαρμογών όπου η διαθεσιμότητα της πληροφορίας είναι καίριας σημασίας, όπως το e-commerce, θα ακολουθηθούν πρακτικές όπου θα δημιουργούνται τοπικά αντίγραφα των πληροφοριών ανά γεωγραφική περιοχή. Τα αντίγραφα αυτά θα πρέπει να ενημερώνονται συστηματικά και το σύστημα θα πρέπει να είναι σε θέση να αντιμετωπίσει τις κατά τόπου καταστροφές. Επίσης η αύξηση του διαθέσιμου bandwith στο Διαδίκτυο κάθε χρόνο κατά 300% κανεί δυνατή την υιοθέτισή του Διαδικτύου ως το εσωτερικό δίκτυο μιας εταιρίας.

- **Αυξημένες ανάγκες ασφάλειας πρόσβασης** Οι εφαρμογές που περιγράψαμε παραπάνω έχουν αυξημένες απαιτήσεις ασφάλειας λόγω της φύσης των συναλλαγών που πραγματοποιούνται. Το πρόβλημα είναι ότι τα δίκτυα διασύνδεσης για τα συστήματα SAN έχουν προκύψει ως δικτυακή επέκταση υφισταμένων εσωτερικών μηχανισμών των συστημάτων αποθήκευσης, και οι παράμετροι ασφάλειας είναι περιορισμένες ως ανύπαρκτες. Θα μπορούσαμε να προκρίνουμε ως λύση την αποτροπή της εξωτερικής επικοινωνίας με τη "φυσική" απομόνωση του συστήματός μας, γεγονός όμως που θα περιορίσει την επεκτασιμότητά του.



Σχήμα 1.8: Λειτουργία και δικτυακές συνδέσεις διαφόρων περιπτώσεων NAS ή SAN.

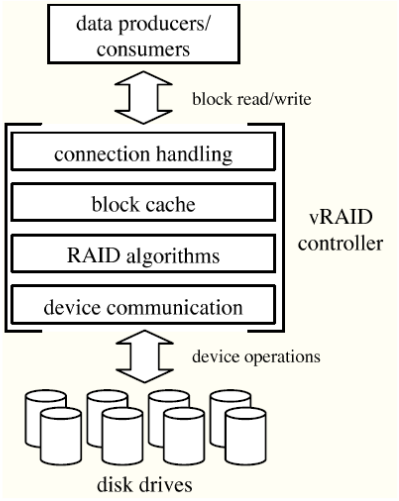
- **Αύξηση επίδοσης ανάλογη της χωρητικότητας** Η επίδοση του συστήματός μας, είτε αυτή μετράται ως προσπελάσεις είτε ως megabytes το δευτερόλεπτο, είτε ως εντολές I/O το δευτερόλεπτο, θα πρέπει να αυξάνεται ανάλογα της χωρητικότητας, έτσι ώστε να μπορέσει να αντιμετωπίσει τον αυξανόμενο αριθμό των πελατών αλλά και των απαιτήσεων των εφαρμογών.

Για να μπορέσουν να γίνουν εφικτοί οι παραπάνω στόχοι χρειάζεται οι δύο υφιστάμενες τεχνολογίες να συγκλίνουν.

1.7 vRAID

Το vRAID [6] είναι ένα κατανεμημένο σύστημα αποθήκευσης που αναπτύσσεται αυτή τη στιγμή από το εργαστήριο. Ανήκει στην κατηγορία των SAN προσφέροντας ως επίπεδο επικοινωνίας αυτό του block. Σκοπός του είναι να μπορέσει να προσφέρει ένα ενιαίο block device, του οποίου το μέγεθος να μπορεί να αυξομειώνεται κατά τις απαιτήσεις της εφαρμογής. Παράλληλα θα προσφέρει αυξημένες δυνατότητες διαχείρισης επιτρέποντας την εύκολη δημιουργία αντιγράφων ασφαλείας. Στο επίπεδο του hardware προσπαθεί να εκμεταλλευτεί ήδη δημοφιλείς λύσεις, όπως το Ethernet στο επίπεδο δικτύωσης, ενσωματώνοντας όμως και ποιο εξελιγμένες λύσεις hardware στα επιμέρους τμήματά του που επιτρέπουν και την αύξηση των επιδόσεών του. Έτσι θα είναι εύκολη η εγκατάστασή του στις υπάρχουσες υποδομές χωρίς αύξηση του κόστους, ενώ παράλληλα θα είναι δυνατή και η μετάβαση σε μελλοντικές λύσεις.

1.7.0.1 Αρχιτεκτονική



Σχήμα 1.9: Η αρχιτεκτονική του συστήματος vRAID

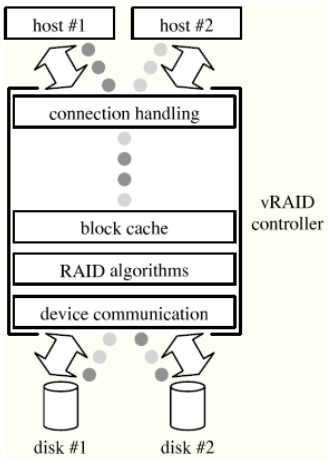
Η αρχιτεκτονική του συστήματος φαίνεται στο σχήμα 1.9. Μπορούμε να διακρίνουμε τρία βασικά κομμάτια, με το πρώτο να είναι οι πελάτες του συστήματος, το δεύτερο ο κεντρικός ελεγκτής - controller, και το τρίτο τα απομακρυσμένα αποθηκευτικά συστήματα.

Επίσης μπορούμε να παρατηρήσουμε ότι έχουμε στην ουσία δύο κομμάτια δικτυακής επικοινωνίας. Το πρώτο είναι ανάμεσα στους πελάτες και τον controller, όπου υλοποιούνται διεπαφές επικοινωνίας που υποστηρίζονται από τα υπάρχοντα λειτουργικά συστήματα, το Linux και το FreeBSD, όπως το NDB και το Ggate ή το AoE, και περιγράφονται αναλυτικά στην παράγραφο 4.1. Οι παραπάνω διεπαφές κάνουν χρήση της υπάρχουσας υποδομής δικτύωσης Ethernet, περιορίζοντας σημαντικά το κόστος υλοποίησης, αφού δεν απαιτούν την ύπαρξη εξειδικευμένης τεχνολογίας δικτύωσης.

Το δεύτερο κομμάτι δικτυακής επικοινωνίας είναι ανάμεσα στον controller και τα απομακρυσμένα συστήματα αποθήκευσης. Επειδή το κομμάτι αυτό της δικτύωσης είναι αρκετά περιορισμένο, αλλά και σημαντικό για την απόδοση του συστήματος, στο παρόν κείμενο μελετούνται περισσότερες από μια διαθέσιμες τεχνολογίες δικτύωσης. Έτσι όπως θα δούμε στη παράγραφο 4.3 υπάρχει μια υπάρχουσα υλοποίηση με την χρήση του TCP/IP. Σκοπός της υπάρχουσας διπλωματικής είναι να μελετήσει την επίδοση υλοποιήσεων του πρωτοκόλλου επικοινωνίας με την χρήση πακέτων Ethernet, και με την χρήση του συστήματος δικτύωσης Myrinet.

Η εσωτερική δομή του controller φαίνεται καθαρά στο σχήμα 1.9. Απο-

τελείται από τα επίπεδα: της διαχείρισης των συνδέσεων, της block cache, των αλγορίθμων RAID και της επικοινωνίας με τα απομακρυσμένα συστήματα αποθήκευσης. Ο σκοπός του controller είναι πραγματοποιεί τον απαραίτητο έλεγχο ανάμεσα στους πελάτες και τις αποθηκευτικές συσκευές. Λαμβάνει της αιτήσεις I/O από τους πελάτες οι οποίες αναφέρονται στον ενιαίο "εικονικό" χώρο block που παρέχει το σύστημα, και τις μεταφράζει σε αιτήσεις I/O στις επιμέρους αποθηκευτικές συσκευές.

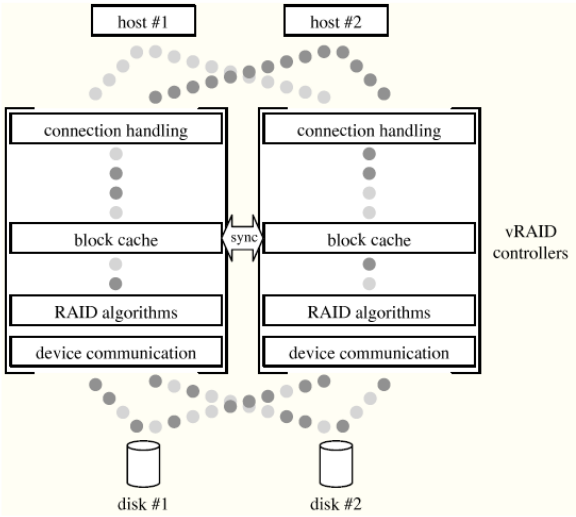


Σχήμα 1.10: Το επίπεδο διαχείρησης συνδέσεων αποκρύπτει από τους πελάτες πως δρομολογούνται οι απαντήσεις από τα χαμηλότερα επίπεδα του controller

1.7.0.1.1 Διαχείριση Συνδέσεων Τοποθετώντας τη διαχείριση συνδέσεων σ' ένα ξεχωριστό επίπεδο οι vRAID controller είναι σε θέση να διαχειριστούν ταυτόχρονα έναν μεγάλο αριθμό από πελάτες. Το επίπεδο αυτό είναι υπεύθυνο για την υλοποίηση του πρωτοκόλλου επικοινωνίας ανάμεσα στους πελάτες και τον controller, εγκαθιστώντας τις ανάλογες συνδέσεις, και αποκρύπτοντας από τα χαμηλότερα επίπεδα του vRAID την πηγή της κάθε αίτησης. Τα χαμηλότερα επίπεδα γνωρίζουν μόνο τα χαρακτηριστικά της κάθε αίτησης και δεν ενδιαφέρονται για την πηγή της αίτησης. Το επίπεδο αυτό σημαδεύει κάθε καινούργια αίτηση μ' έναν αριθμό μοναδικό για κάθε πελάτη. Οι απαντήσεις μπορούν να δρομολογηθούν πάλι πίσω στον αιτούντα. Αυτός ο αριθμός συνοδεύει την αίτηση σε όλο τον κύκλο ζωής της μέσα στον controller, και ανάλογα με την υλοποίηση μπορεί να περιλαμβάνεται και στις ανταλλασσόμενες πληροφορίες με τους απομακρυσμένους δίσκους.

1.7.0.1.2 Block Cache Η διαδικασία της ενδιάμεσης αποθήκευσης χρησιμοποιείται ως μια τεχνική για την επιτάχυνση των αιτήσεων που απευθύνονται

στις ίδιες διευθύνσεις block. Οι vRAID controllerw αποθηκεύουν blocks στην cache στο επίπεδο του ενιαίου αποθηκευτικού χώρου διευθύνσεων, με σκοπό τα δεδομένα να μην φτάσουν στο επίπεδο του RAID. Η καθυστέρηση που αφορά την εγγραφή και την ανάγνωση στο επίπεδο RAID εξαρτάται από τον τύπο της αίτησης, αλλά γενικά απαιτεί την παραγωγή και την εκτέλεση πολυάριθμων αιτήσεων προς τους δίσκους που είναι αρκετά χρονοβόρο.



Σχήμα 1.11: Στην περίπτωση λειτουργίας περισσότερων του ενός controller τότε θα πρέπει να εφαρμοστούν μηχανισμοί συγχρονισμού των περιεχομένων των caches.

Επίσης η ύπαρξη της cache μπορεί να βοηθήσει την επίδοση κατανεμημένων εφαρμογών στους πελάτες οι οποίες απαιτούν την απενεργοποίηση των τοπικών caches για λόγους ακεραιότητας των δεδομένων, γιατί δεν υπάρχει κάποιος μηχανισμός διατήρησης της ακεραιότητας ανάμεσα στους πελάτες. Βέβαια το ίδιο πρόβλημα προκύπτει και για το σύστημα vRAID στην περίπτωση που ενεργοποιηθούν περισσότεροι από ένας controllers (σχήμα 1.11) και επιβάλλεται η χρησιμοποίηση κάποιου πρωτοκόλλου διατήρησης της ακεραιότητας των δεδομένων. Βέβαια η ενεργοποίηση περισσοτέρων του ενός controller επιτρέπουν την αύξηση του φορτίου στο οποίο μπορεί να αντεπεξέλθει το όλο σύστημα, καθώς και την αύξηση της αντοχής του σε περίπτωση που κάποιος από τους controller αποτύχει κατά την λειτουργία του.

1.7.0.1.3 Αλγόριθμοι RAID Σκοπός του συστήματος είναι να υλοποιήσει τους υπάρχοντες αλγορίθμους που χρησιμοποιούνται ως επί το πλείστον από τα περισσότερα αποθηκευτικά συστήματα. Μέχρι στιγμής προσφέρονται

υλοποιήσεις των RAID 0 και RAID 1, με σκοπό στο μέλλον να προστεθούν και υλοποιήσεις των RAID 5 και 6.

1.7.0.1.4 Επικοινωνία με απομακρυσμένες συσκευές Το χαμηλότερο επίπεδο του controller είναι αυτό της επικοινωνίας με τις απομακρυσμένες αποθηκευτικές συσκευές, όπως το περιγράψαμε προηγουμένως.

Κεφάλαιο 2

Περιγραφή Λειτουργίας Πρωτοκόλλων

2.1 Myrinet

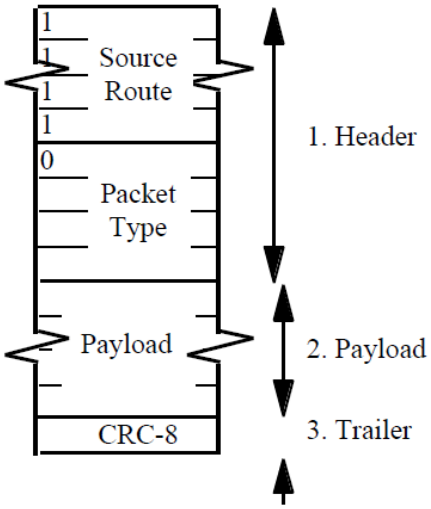
Το Myrinet είναι ένας καινούργιος τύπος τοπικού δικτύου βασισμένο στην τεχνολογία που χρησιμοποιείται για την διασύνδεση συστημάτων MMP (Massively Parallel Processors). Είναι αποτέλεσμα [5] [20] δύο ερευνητικών προγραμμάτων του Mosaic στο Caltech και του ATOMIC LAN του USC [19]. Προσφέρει ταχύτητες μετάδοσης της τάξης των 1.28 Gbit/sec, των 2Gbit/sec και αισίως και 10Gbit/sec. Τα χαρακτηριστικά των δικτύων MPP είναι:

- Οι υψηλοί ρυθμοί μετάδοσης
- Οι κανονικές τοπολογίες και η βαθμωτή επεκτασιμότητα (scalability), όπου συνδέονται στοιχειώδη στοιχεία δρομολόγησης κατά κανονικό τρόπο και η μέγιστη χωρητικότητα του δικτύου αυξάνεται ανάλογα της αύξησης των κόμβων που προστίθενται στο δίκτυο.
- Έχουν χαμηλούς ρυθμούς σφάλματος.
- Υποστηρίζουν δρομολόγηση cut-through. Επειδή η επικοινωνία είναι ιδιαίτερα ασφαλής δεν χρησιμοποιείται δρομολόγηση store-and-forward, όπου λαμβάνεται ολόκληρο το πακέτο, αποθηκεύεται στον δρομολογητή και γίνεται έλεγχος ακεραιότητάς και στην συνέχεια στέλνεται στο άλλο άκρο. Αντίθετα το πακέτο εξέρχεται από τον δρομολογητή μόλις ληφθεί και αποκωδικοποιηθεί η επικεφαλίδα του.
- Υποστήριξη ελέγχου ροής σε κάθε σύνδεσμο επικοινωνίας. Σε περίπτωση που το κανάλι εξόδου είναι κατειλημμένο στη μέθοδο δρομολόγησης cut-through το πακέτο διακόπτεται με την χρήση ελέγχου ροής στο κανάλι

επικοινωνίας, και έτσι δεν χρειάζεται να γίνεται ενδιάμεση αποθήκευση των μηνυμάτων. Ο έλεγχος ροής γίνεται με την επιβεβαίωση κάθε στοιχειώδους μονάδας, ενός byte δηλαδή.

2.1.1 Μορφή πακέτου Myrinet

Η μορφή του πακέτου Myrinet είναι αυτή που φαίνεται στο σχήμα 2.1 και η τυπική τιμή του MTU (Maximun Transmission Unit) μπορεί να είναι 4Mbytes. Το πακέτο αποτελείται από τα εξής κομμάτια:



Σχήμα 2.1: Μορφής ενός πακέτου Myrinet

1. Την επικεφαλίδα που μπορεί να έχει μέγεθος τέσσερα ή περισσότερα bytes.
2. Το Payload που μπορεί να έχει μέγεθος μηδέν ή περισσότερα bytes.
3. Το Trailer που μπορεί να έχει μέγεθος ένα byte.

2.1.1.1 Επικεφαλίδα

Η επικεφαλίδα του πακέτου μπορεί να περιλαμβάνει κατά σειρά τα παρακάτω:

1. **Source Route** Είναι ένα σύνολο από μηδέν ή περισσότερα bytes που υποδηλώνουν την διαδρομή που πρέπει να ακολουθήσει το πακέτο για να μπορέσει να φτάσει στο προορισμό του. Για όλα τα byte αυτής της ομάδας θα πρέπει το MSB (Most Signigicant Bit) να έχει την τιμή 1.

2. **Packet Type** Το μέγεθος αυτού του πεδίου μπορεί να είναι 4 bytes. Η τιμές του αντιπροσωπεύουν το πρωτόκολλο και το αντίστοιχο software που χρειάζεται να χρησιμοποιηθεί για τον χειρισμό του εισερχόμενου πακέτου. Οι διάφοροι υποστηριζόμενοι τύποι μπορούν να βρεθούν στην παρακάτω διεύθυνση http://www.myri.com/scs/types.html

2.1.1.2 Packet Payload

Το περιεχόμενό του καθορίζεται από τα πρωτόκολλα που βρίσκονται υψηλότερα από το επίπεδο του Data Link. Το Myrinet έχει την δυνατότητα να μεταφέρει πακέτα Ethernet και έτσι μπορεί εύκολα να μεταδώσει όσα πρωτόκολλα υποστηρίζονται από το Ethernet.

2.1.1.3 Packet Trailer

Το τελευταίο byte είναι το CRC-8 (Cyclic-Redundancy-check) όλων των προηγούμενων δεδομένων που βρίσκονται στο πακέτο συμπεριλαμβανομένης και της επικεφαλίδας, χρησιμοποιώντας το πολυώνυμο X^8+X^2+X+1.. Επειδή το πεδίο του Source Route αλλάζει κατά την διέλευσή του από το switch θα πρέπει να ξαναϋπολογιστεί για κάθε link. Κατά την είσοδο του πακέτου σε κάποιοι switch ή interface γίνεται ο υπολογισμός τους CRC-8 και γίνεται η πράξη του XOR ανάμεσα στην υπολογισμένη τιμή και την τιμή που υπάρχει στο πεδίο Trailer. Αν το CRC-8 είναι σωστό τότε η τιμή που θα βρίσκεται στο Trailer θα είναι μηδενική. Κατά την έξοδο του πακέτου από το switch επαναϋπολογίζεται το CRC-8, και πραγματοποιείται η πράξη του XOR με την τιμή του Trailer. Αν είχε την σωστή τιμή CRC όταν λάβαμε το πακέτο τότε θα έχει και την σωστή τιμή CRC όταν θα το στείλουμε.

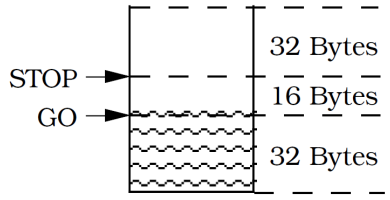
2.1.2 Σύνολο χαρακτήρων

Το σύνολο χαρακτήρων του Myrinet περιλαμβάνει τις 256 διαφορετικές τιμές που μπορεί να πάρει ένα byte και κάποιους επιπλέον χαρακτήρες ελέγχου. Αυτοί είναι:

- **GAP** Αυτός ο χαρακτήρας ελέγχου χρησιμοποιείται για να υποδηλώσει τον τέλος ενός πακέτου.
- **STOP και GO** Χρησιμοποιούνται για την υλοποίηση του ελέγχου ροής.
- **non-IDLE** Ο αποστολέας είναι υποχρεωμένος να στέλνει αυτήν την εντολή ανά τακτά χρονικά διαστήματα για να είναι σε θέση να εντοπίζει τις ανοικτές συνδέσεις.

2.1.3 Έλεγχος Ροής

Όπως είδαμε παραπάνω ο έλεγχος ροής πραγματοποιείται με την χρήση των χαρακτήρων ελέγχου STOP και GO που στέλνει ο παραλήπτης.



Σχήμα 2.2: Μορφή του slack buffer

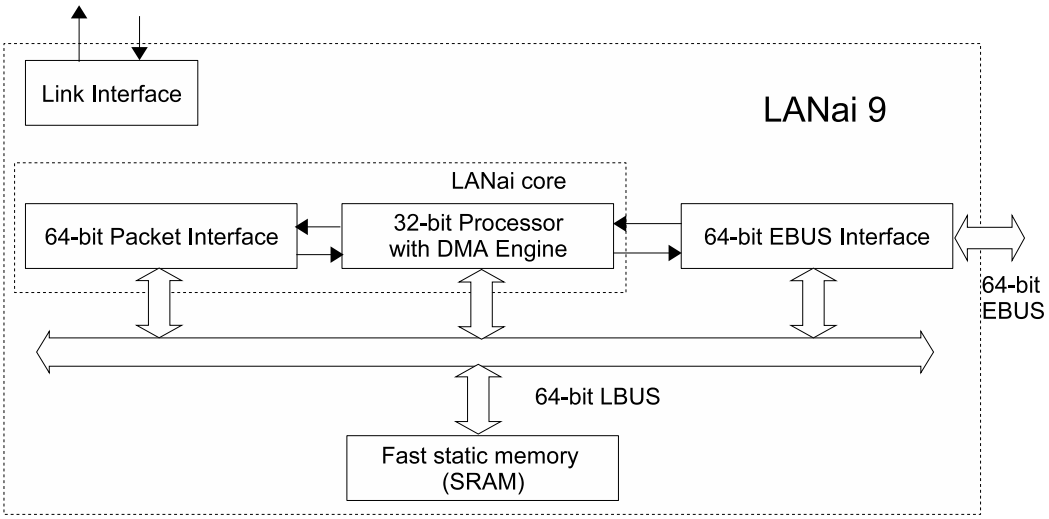
Ο έλεγχος ροής υλοποιείται με την βοήθεια ενός slack buffer όπως αυτός φαίνεται στο σχήμα 2.2. Αν η ροή εξόδου έχει μπλοκαριστεί έτσι ώστε να φτάσουμε στον δείκτη STOP τότε ο δέκτης αποστέλλει την ομόνιμη εντολή. Όταν αντίστοιχα έχουμε έναρξη της ροής εξόδου και ο δείκτης φτάσει κάτω από το όριο GO τότε στέλνεται η ανάλογη εντολή και ξεκινά εκ νέου η λήψη δεδομένων.

2.1.4 Δρομολόγηση

Στη δρομολόγηση το Myrinet βασίζεται σε crossbar switches [15] που είναι κατάλληλα τοποθετημένοι έτσι έστω ο γράφος που δημιουργείται να έχει την μεγαλύτερη δυνατή ελάχιστη διχοτόμηση (bisection). Επίσης είδαμε ότι η τεχνική δρομολόγησης είναι η cut-through δηλαδή το πακέτο προωθείται στον προορισμό του μόλις αποκωδικοποιηθεί η διεύθυνση προορισμού. Τέλος η μέθοδος δρομολόγησης που ακολουθείται είναι αυτή της δρομολόγησης πηγής (source routing). Είναι στην ευθύνη του κάρτας Myrinet κατά την αποστολή του πακέτου να έχει καθορίσει την διαδρομή που αυτό θα ακολουθήσει. Κατά το πέρασμά του από τις διάφορες πόρτες αφαιρείται και η αντίστοιχη διεύθυνση από την επικεφαλίδα του πακέτου. Το πεδίο της διεύθυνσης ερμηνεύεται από τα switch ως η διαφορά της πόρτας εξόδου από την πόρτα εισόδου. Το πλεονέκτημα αυτής της μεθόδου είναι ότι μας επιτρέπει να βρούμε εύκολα την ανάποδη διαδρομή.

2.1.5 Προσαρμογέας Δικτύου

Το block διάγραμμα μιας κάρτα Myrinet φαίνεται στο σχήμα. 2.3. Το μέγεθος της SRAM μπορεί να φτάσει μέχρι και τα 16Mbytes, με τις τυπικές τιμές της να είναι 2 ή 4 Mbytes. Χρησιμοποιείται για την αποθήκευση τους



Σχήμα 2.3: Block διάγραμμα του προσαρμογέας δικτύου Myrinet

Myrinet Control Program (MCP) και για την ενδιάμεση αποθήκευση των πακέτων Στην καρδία του βρίσκεται ένα RISC επεξεργαστής που αισίως έχει φτάσει στην ταχύτητα των 333MHz. Σημαντική παράμετρος στην επίδοση του συστήματος είναι η ύπαρξη μηχανών DMA που επιτρέπουν την απευθείας μεταφορά δεδομένων που βρίσκονται στην κεντρική μνήμη στην μνήμη της κάρτας.

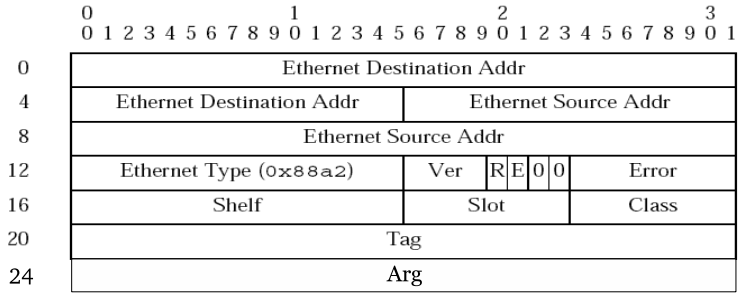
2.2 AoE

Το πρωτόκολλο Ata over Ethernet είναι μια προσπάθεια της εταιρίας Coraid [22] για τη δημιουργία ενός πρωτοκόλλου που θα επιτρέπει την γρήγορη μεταφορά δεδομένων από έναν δικτυακό σκληρό δίσκο σ' ένα κεντρικό σύστημα με ασφαλή τρόπο.

Η ιδέα πίσω από την δημιουργία του όλου συστήματος είναι ότι πρέπει να βασίζεται σε υπάρχοντα φτηνό εξοπλισμό που θα είναι ευρέως διαδεδομένος, και σκοπός είναι να μεγιστοποιηθεί η απόδοση ενός τέτοιου συστήματος. Από μόνες τους οι δύο αυτές παράμετροι καταδεικνύουν ότι η χρήση του Ethernet ως βάσης για το επίπεδο δικτύωσης είναι μονόδρομος, αφού χρησιμοποιείται παντού, και ο εξοπλισμός είναι πάμφθηνος.

Η δεύτερη παράμετρος του προβλήματος είναι να μεγιστοποιηθεί η απόδοση του όλου συστήματος, δηλαδή την επικοινωνία σκληρού δίσκου με το δίκτυο στον server και δικτύου με το επίπεδο του linux block device στον πελάτη. Γενικά το σύστημα απευθύνεται σε λύσεις τοπικής αποθήκευσης. Ο server και ο πελάτης βρίσκονται τοπικά στον ίδιο χώρο και η επικοινωνία τους είναι δυνατή μόνο με την χρήση ενός Ethernet switch. Το πρωτόκολλο που πρέπει να δημιουργηθεί για την επικοινωνία όχι μόνο δε πρέπει να απευθύνεται σε περιπτώσεις απομακρυσμένης επικοινωνίας, αλλά και να απαλλαγεί από επιπλέον επίπεδα επικοινωνίας που αποδεικνύονται άχρηστα. Έτσι όλοι οι μηχανισμοί που προσφέρει ο συνδυασμός του TCP/IP και περίπλοκοι είναι για την εφαρμογή αυτή, και εισάγουν και επιπλέον επεξεργαστικό φόρτο που δε χρειάζεται. Καταλήγουμε στο συμπέρασμα ότι η συγγραφή του πρωτοκόλλου αυτού ακριβώς πάνω από το Ethernet επίπεδο, είναι η καλύτερη λύση, γιατί η επικοινωνία σε τοπικό επίπεδο είναι εφικτή και απαλλασσόμαστε από μηχανισμούς ελέγχου που πλέον η εφαρμογή μας δεν χρειάζεται. Για την εξασφάλιση της σίγουρης μετάδοσης των μηνυμάτων το AoE κάνει χρήση απλών μηχανισμών αναμετάδοσης των πακέτων για τα οποία δεν έχουμε λάβει απάντηση μετά από κάποιο χρονικό διάστημα.

Ας δούμε τώρα την πλευρά του εξυπηρετητή (server). Σκοπός εδώ είναι να βελτιστοποιήσουμε την επικοινωνία μεταξύ του δίσκου και του δικτύου. Σ' ένα τυπικό υπολογιστή, η απόσταση μεταξύ του δικτύου και του σκληρού είναι σχετικά μεγάλη καθώς παρεμβάλλονται πολλά επίπεδα όχι μόνο hardware αλλά και software. Η προσέγγιση που ακολούθησαν στην Coraid ήταν να δημιουργήσουν μια embedded συσκευή που στην ουσία αποτελείται από έναν ελεγκτή δίσκου και μια κάρτα δικτύου που σε συνδυασμό με κατάλληλα λογικά κυκλώματα επιτρέπει την επεξεργασία των πακέτων του πρωτοκόλλου.



Σχήμα 2.4: Επικεφαλίδα ενός μηνύματος AoE

2.2.1 Περιγραφή του πρωτοκόλλου

Το πρωτόκολλο Ata over Ethernet (AoE) είναι στην ουσία του ένα πολύ ελαφρύ πρωτόκολλο, προσαρμοσμένο στα παρεχόμενα εργαλεία και τα χαρακτηριστικά του προβλήματος. Η περιγραφή του πρωτοκόλλου βρίσκεται στα παρακάτω κείμενα [24, 23] Στο σχήμα 2.4 μπορείτε να δείτε την κοινή επικεφαλίδα των πακέτων του AoE.

2.2.1.1 Επικεφαλίδα του πρωτοκόλλου

Τα πεδία της επικεφαλίδας είναι τα παρακάτω:

- **Ethernet επικεφαλίδα:** Περιλαμβάνει τις MAC διευθύνσεις του παραλήπτη και του αποστολέα καθώς και τον κωδικό Ethernet του πρωτοκόλλου (Ethernet Type). Θα μπορούσε να παραλείπεται αφού ανήκει σε άλλο επίπεδο δικτύωσης, αλλά για την πληρότητα της απεικόνισης περιλαμβάνεται και εδώ. Το πεδίο Ethernet Type χρησιμοποιείται για να μπορούμε να ξεχωρίσουμε τα πρωτόκολλα που είναι γραμμένα πάνω από το Ethernet.

- **Αριθμός Έκδοσης:** Ο αριθμός έκδοσης (version number) καθορίζει τη μορφή της επικεφαλίδας του AoE, καθώς επίσης και το σύνολο τον κωδικών εντολής. Όλες οι μελλοντικές εκδόσεις πρέπει να περιλαμβάνουν αυτό το πεδίο στη συγκεκριμένη θέση

- **Bit Ελέγχου - Flags:** Αυτά τα 4 bit έχουν τα παρακάτω πεδία ονομάζονται κατά σειρά R, E ,Z και Z. Το R bit γίνεται ένα όταν το μήνυμα είναι απάντηση από τον εξυπηρετητή. Σκοπός του είναι οι εξυπηρετητές να απορρίπτουν αμέσως απαντήσεις που εσφαλμένα έφτασαν σε αυτούς. Το E bit γίνεται ένα όταν η απάντηση περιέχει κάποιον κωδικό σφάλματος. Τα Z bits είναι δεσμευμένα και πρέπει να είναι πάντα 0.

- **Κωδικός Σφάλματος:** Όταν το bit E είναι 1 τότε το πεδίο Error μπορεί να περιέχει έναν από τους ακόλουθους κωδικούς σφάλματος.

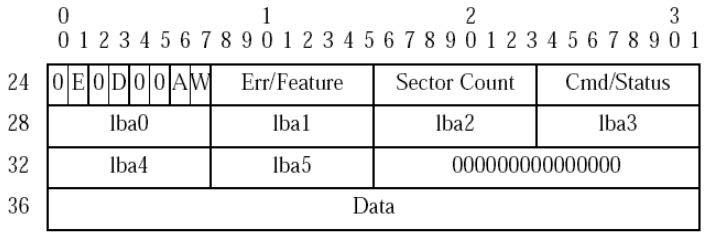
  - Σφάλμα 1 : Άγνωστος κωδικός εντολής. Ο εξυπηρετητής δεν αναγνωρίζει τον κωδικό εντολής του αντίστοιχου πεδίου.
  - Σφάλμα 2 : Εσφαλμένη τιμή παραμέτρου. Υπάρχει εσφαλμένη τιμή σε κάποιο από τα πεδία που περιέχονται στο πεδίο Arg.
  - Σφάλμα 3 : Συσκευή ανενεργή. Ο εξυπηρετητής δε μπορεί να δεχτεί πλέον άλλες εντολές ATA.
  - Σφάλμα 4 : Υπάρχουσα συμβολοακολουθία ελέγχου. Η συμβολοακολουθία ελέγχου (configuration string) δε μπορεί να ανατεθεί, γιατί προυπάρχει τιμή.
  - Σφάλμα 5 : Μη υποστηριζόμενη έκδοση. Ο εξυπηρετητής δεν είναι σε θέση να απαντήσει στο εν λόγω μήνυμα, επειδή δεν υποστηρίζει την έκδοση του πρωτοκόλλου.

- **Shelf** [1] **Slot** [2]**:** Το Shelf αναφέρεται στον αριθμό του rack που βρίσκονται τοποθετημένοι οι εξυπηρετητές Το Slot είναι ο αριθμός του εξυπηρετητή μέσα στο δεδομένο Shelf. Σκοπός των δύο αυτών αριθμών είναι να διευθυνσιοδοτήσουν τους εξυπηρετητές βάσει της φυσικής τους θέσης, και να απαντάνε μόνο σε μηνύματα που απευθύνονται σε αυτούς. Επίσης ένας εξυπηρετητής θα πρέπει να απαντήσει αν το πεδίο Shelf είναι το δικό του ή έχει την τιμή 0xffff και όταν το πεδίο Slot έχει την δική του τιμή ή έχει την τιμή 0xff. Στις απαντήσεις του ο εξυπηρετητής θα πρέπει να βάζει πάντα τους δικούς του αριθμούς Shelf και Slot.

- **Εντολή - Class:** Το πεδίο εντολή (Class ή Command) δηλώνει τον τύπο της εντολής. Οι τιμές που μπορεί να πάρει είναι οι: 0 και αντιστοιχεί σε εντολή ATA και 1 που αντιστοιχεί σε μύνημα ρύθμισης - ελέγχου.

- **Αριθμός Ακολουθίας - Tag:** Το πεδίο αυτό περιέχει τον αριθμό ακολουθίας του μηνύματος και επιτρέπει έτσι στον πελάτη να αντιστοιχίζει τις αιτήσεις με τις απαντήσεις του. Κάθε απάντηση θα πρέπει να αντιγράφει τον αριθμό ακολουθίας της αίτησης.

- **Arg:** Τα περιεχόμενα αυτού του πεδίου είναι η είσοδος για την εντολή ελέγχου.



Σχήμα 2.5: Μορφή πακέτου εντολής ATA

2.2.1.2 Επικεφαλίδα εντολής ATA

Όπως έχουμε αναφέρει προηγουμένως το πρωτόκολλο AoE έχει δύο τύπους μηνυμάτων, όπου ο ένας από αυτούς είναι υπεύθυνος για την μεταφορά των εντολών ATA. Όπως είδαμε κατά την σύντομη περιγραφή του πρωτοκόλλου ATA η έκτη έκδοση του επιτρέπει την αύξηση των διευθυνσιοδοτούμενων sectors, διπλασιάζοντας στην ουσία τους διαθέσιμους καταχωρητές για τις εντολές ATA, και ολοκληρώνοντας την αποστολή τους σε δύο κύκλους. Το πρωτόκολλο με την σειρά του περιλαμβάνει όλη αυτήν την πληροφορία σε ένα πακέτο όπως αυτό φαίνεται στο σχήμα 2.5. Ο κωδικός της εντολής είναι 0. Το μέγεθος του όλου μηνύματος δε μπορεί να υπερβαίνει μαζί με την επικεφαλίδα Ethernet τα 1520 bytes, οπότε είναι προφανές ότι ο μέγιστος αριθμός των sectors που μπορούμε να μεταφέρουμε σε κάθε πακέτο περιορίζεται στους 2.

Η πρώτη οκτάδα από bits έχει ρυθμιστικό ρόλο για το είδος του μηνύματος και επιβάλλει και την ανάλογη συμπεριφορά του server. Έτσι το bit W έχει την τιμή 1 όταν η εντολή ATA απαιτεί να γίνει κάποια εγγραφή στη συσκευή. Το bit A πρέπει να έχει την τιμή 1 όταν θέλουμε η εγγραφή να γίνει ασύγχρονα. Το D bit έχει την ίδια σημασία όπως είναι καθορισμένη στο καταχωρητή Device/Head καταχωρητή του ATA, υποδηλώνει δηλαδή ποια συσκευή είναι επιλεγμένη, και αποτιμάται μόνο αν το E bit έχει τιμή 1. Το E bit παίρνει την τιμή 1 αν το μήνυμα είναι μια LBA48 εντολή.

Το W bit σε συνδυασμό με το πεδίο Sector Count καθορίζουν την συμπεριφορά της εντολής, αν θα γίνει δηλαδή ανάγνωση ή εγγραφή. Έτσι μπορούμε να ξεχωρίσουμε τις παρακάτω περιπτώσεις.

- Αν πρέπει να εγγραφούν δεδομένα τότε το W bit πρέπει να έχει την τιμή 1 και το πεδίο Data πρέπει να περιέχει Sector count * 512 bits δεδομένων.
- Αν πρέπει να αναγνωστούν δεδομένα από τη συσκευή τότε θα πρέπει το

[1]Το πεδίο αυτό σε άλλα σημεία της περιγραφής αναφέρεται ως Major
[2]Το πεδίο αυτό σε άλλα σημεία της περιγραφής αναφέρεται ως Minor

W bit να έχει την τιμή 0 και ο το πεδίο Sector Count να περιέχει τον αριθμό των sectors που πρέπει να αναγνωστούν από το σύστημα. Αν η εντολή είναι επιτυχής τότε το πεδίο Data της απάντησης θα περιέχει τα ανεγνωσμένα δεδομένα.

- Αν δε πρέπει να μεταφερθούν δεδομένα τότε θα πρέπει ο Sector Count να έχει την τιμή 0 και η τιμή του W αγνοείται.

Αν και τα δύο bits A, W έχουν την τιμή 1, τότε ο server μπορεί να αποθηκεύσει προσωρινά την εντολή στην μνήμη του και να απαντήσει άμεσα, επιστρέφοντας το Arg πεδίο αναλύωτο. Ο server μπορεί να εκτελέσει την εντολή όποτε εκείνος θεωρεί αναγκαίο, υπό την προϋπόθεση ότι κάποια επόμενη εντολή read θα επιστρέψει τα προσωρινά αποθηκευμένα δεδομένα. Αν ο server σταματήσει να λειτουργεί λόγω βλάβης τότε τα δεδομένα που βρίσκονται στην cache του μπορεί να χαθούν. Καμία απάντηση δεν αποστέλλεται όταν ολοκληρωθεί η εγγραφή ακόμα και όταν προκύψει κάποιο σφάλμα.

Αν το E bit έχει την τιμή 1 τότε τα περιεχόμενα του πακέτου θα μεταφερθούν στους καταχωρητές της συσκευής ATA με την ακόλουθη σειρά.

Συσκευή ATA	Πακέτο AoE
Device	(AFlags & 0x50 || 0xA0)
LBA Low	lba3
LBA Low	lba0
LBA Mid	lba4
LBA Mid	lba1
LBA High	lba5
LBA High	lba2
Sector Count	0
Sector Count	Sector Count
Err/Feature	Err/Feature
Cmd/Status	Cmd/Status

Τα bits E,D αντιστοιχούν στα bit LBA και DEV του καταχωρητή Device του ATA.. Η πράξη OR, που φαίνεται στον παραπάνω πίνακα, γίνεται έτσι ώστε τα obsolete bits να πάρουν την τιμή 1 όπως περιγράφεται στο πρότυπο ATA.

Αν το E bit έχει την τιμή 0, τότε η εντολή είναι της μορφής LBA24 και οι καταχωρητές λαμβάνουν τις παρακάτω τιμές.

Συσκευή ATA	Πακέτο AoE
Device	lba3
LBA Low	lba0
LBA Mid	lba1
LBA High	lba2
Sector Count	Sector Count
Err/Feature	Err/Feature
Cmd/Status	Cmd/Status

Εκτός από την περίπτωση των ασύγχρονων εγγραφών, απάντηση δεν παράγεται παρά μόνο την ολοκλήρωσης της εντολής, είτε αυτή είναι επιτυχημένη ή όχι. Μετά το τέλος της εντολής τα περιεχόμενα των καταχωρητών ATA αντιγράφονται στα πεδία του πακέτου ως ακολούθως, ανεξαρτήτως της τιμής του bit E:

Πακέτο AoE	Συσκευή ATA
Err/Feature	Err/Feature
Sector Count	Sector Count
lba0	LBA Low
lba1	LBA Mid
lba2 Count	LBA High
Cmd/Status	Cmd/Status

Στη περίπτωση μάλιστα που το E bit έχει την τιμή 1, που σημαίνει ότι είχαμε μια εντολή της μορφής LBA48, τθέτοντας το bit HOB=1 του καταχωρητή ελέγχου συσκευής (Device Control register σχήμα: 1.3β) θα αναθέσουμε τους υπόλοιπους καταχωρητές ως εξής:

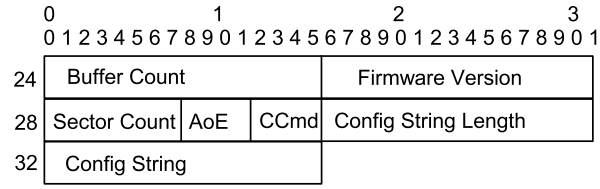
lba3	LBA Low
lba4	LBA Mid
lba5	LBA High

Όλα τα υπόλοιπα πεδία του πακέτου της απάντησης λαμβάνουν τις τιμές που είχε το πακέτο της εντολής.

2.2.1.3 Επικεφαλίδα εντολής Ρύθμισης και Ελέγχου

Τα πακέτα τύπου Ρύθμισης και Ελέγχου (Query Config Information) χρησιμοποιούνται για τη λήψη πληροφοριών από τον server, αλλά και για την ανάθεση τιμών. Η σημασία του κάθε πεδίου περιγράφεται παρακάτω.

- **Buffer Count:** Αναφέρεται στον μέγιστο αριθμό ξεχωριστών μηνυμάτων που ο server είναι σε θέση να έχει σε αναμονή για επεξεργασία. Τα μηνύματα πέραν αυτής της χωρητικότητας απορρίπτονται.



Σχήμα 2.6: Μορφή πακέτου εντολής Ρύθμισης και Ελέγχου

- **Firmware Version:** Η έκδοση του firware του server.
- **Sector Count:** Αν μη μηδενικό , αυτό το πεδίο καθορίζει τον μέγιστο αριθμό sector που ο server μπορεί να διαχειριστεί σε μια εντολή ATA. Αυτό ο αριθμός μπορεί να προκύψει από περιορισμούς του server , της μνήμης του ή του μέγιστου μεγέθους τους πακέτου του δικτύου.
- **AoE:** Η έκδοση του πρωτοκόλλου AoE που υποστηρίζει ο server.
- **CCmd:** Ο αριθμός της υποεντολής που θα εκτελεστεί σε σχέση με τη συμβολοσειρά ελέγχου.
- **Config String Length:** Το μήκος της συμβολοσειράς ελέγχου.
- **Config String:** Η συμβολοσειρά ελέγχου, με μέγιστο μήκος 1024.

Σε μια τέτοια εντολή τα πεδία Buffer Count , Firmware Version και AoE, πρέπει να έχουν τιμή μηδέν από τον πελάτη, και δε θα ληφθούν υπόψη από τον server. Τα υπόλοιπα πεδία μπορούν να χρησιμοποιηθούν για την ανάκληση ή την ανάθεση της συμβολοσειράς ελέγχου.

Οι τιμές που μπορεί να λάβει το πεδίο CCmd μπορεί να είναι:

- **CCmd 0: Ανάγνωση της συμβολοσειράς ελέγχου** Ανάγνωση της συμβολοσειράς ελέγχου του server χωρίς να πραγματοποιήσεις κάποιον έλεγχο.
- **CCmd 1: Έλεγχος της συμβολοσειράς ελέγχου** Απάντησε μόνο αν η συμβολοσειρά του μηνύματος ταιριάζει ακριβώς με τη συμβολοσειρά του server.
- **CCmd 2: Έλεγχος του προθέματος της συμβολοσειράς ελέγχου** Απάντησε μόνο αν τη συμβολοσειρά του μηνύματος είναι πρόθεμα της συμβολοσειράς ελέγχου του server.
- **CCmd 3: Καθορισμός της συμβολοσειράς ελέγχου** Αν η υπάρχουσα συμβολοσειρά ελέγχου του server είναι κενή, τότε ανάθεσε της την

τιμή που περιέχεται στο ανάλογο πεδίο του μηνύματος. Αν δεν είναι κενή τότε στείλε απάντηση με το E bit 1 και το πεδίο Error με τιμή 4.

- **CCmd 4: Βίαιος Καθορισμός της συμβολοσειράς ελέγχου** Ανάθεσε στη συμβολοσειρά ελέγχου την τιμή του πεδίου του μηνύματος και απάντησε.

Στην απάντησή του ο server πρέπει να αναθέτει τις τιμές των τρεχουσών του τιμών

Όταν ένας server ξεκινά και είναι έτοιμος για την επεξεργασία εντολών θα πρέπει να στέλνει ένα Broadcast Ethernet μήνυμα, με περιεχόμενα μια απάντηση ρύθμισης και ελέγχου όπου το πεδίο Tag πρέπει να έχει την τιμή 0.

2.2.2 Περιγραφή Λειτουργίας

Ας δούμε τώρα ανά τμήματα την περιγραφή του πρωτοκόλλου

2.2.2.1 Επικεφαλίδα

Το κοινό κομμάτι όλων των μηνυμάτων είναι αυτό που εξασφαλίζει τους μηχανισμούς της διευθυνσιοδότησης και της ακεραιότητας της λειτουργίας του πρωτοκόλλου. Η ακεραιότητα της λειτουργίας, η εξασφάλιση δηλαδή της εκτέλεσης της εντολής και της απάντησης στον client , γίνεται μέσω του πεδίου TAG που είναι ο αριθμός ακολουθίας της κάθε εντολής. Κατά την απάντησή του ο server αντιγράφει το πεδίο αυτό, και έτσι ο client είναι σε θέση κάνοντας αναζήτηση σε μια λίστα που περιέχει όσα μηνύματα έχει στείλει μέχρι αυτήν τη στιγμή, να αντιστοιχίσει την απάντηση με την αίτηση, και έτσι να εντοπίσει τα πιθανώς χαμένα πακέτα με την χρήση κατάλληλων χρονομέτρων.

Η λειτουργία της διευθυνσιοδότησης όπως είδαμε γίνεται με την χρήση των πεδίων Shelf και Slot. Οι server AoE είναι οργανωμένοι σε racks. Κάθε rack έχει τον δικό του αριθμό Shelf που καθορίζεται κατά την ρύθμιση του. Με την σειρά του κάθε server έχει τον δικό του αριθμό Slot. Έχουμε λοιπόν ένα ζεύγος αριθμών που μας επιτρέπει να βρούμε την ακριβή θέση του κάθε server. Το κρίσιμο σημείο είναι πώς θα βρούμε την mac διεύθυνση του server αν γνωρίζουμε μόνο τους δύο αριθμούς Shelf και Slot. Ο client λοιπόν στέλνει ένα broadcast μήνυμα ελέγχου με τα πεδία Sheft και Slot να περιέχουν τις τιμές του προς αναζήτηση server. Το μήνυμα θα το λάβουν όλοι οι server αλλά θα απαντήσει μόνο αυτός που βρίσκεται στη συγκεκριμένη θέση. Με την ίδια λογική όταν κάποιος server ξεκινήσει θα αποστείλει ένα broadcast μήνυμα, έτσι ώστε να τον ανακαλύψουν οι client.

Επιπλέον το πρωτόκολλο υποστηρίζει broadcast τιμές στα πεδία Slot και Shelf, όταν τα πεδία αυτά έχουν σε όλα τους τα bit την τιμή ένα. Οι server

που απαντούν σε αυτή την περίπτωση είναι αυτοί που η τιμή τους σε ένα από τα δύο πεδία ταυτίζεται με την δική τους. Για παράδειγμα αν αποσταλεί μήνυμα όπου το πεδίο Shelf έχει broadcast τιμή και το Slot έχει την τιμή έξι, τότε θα απαντήσουν όλοι οι server σε κάθε rack που έχουν τιμή Slot ίση με έξι. Αν πάλι το Self έχει τιμή ένα και το Slot έχει broadcast τιμή τότε θα απαντήσουν όλοι οι server που βρίσκονται στο rack με αριθμό ένα.

2.2.2.2 Μηνύματα ATA

Αυτά είναι τα μηνύματα που χρησιμοποιούνται για την μεταφορά των δεδομένων. Στη πλευρά του server γίνεται άμμεση επεξεργασία τους με την χρήση των κυκλωμάτων που διαθέτει η embedded υλοποίηση Στο πλευρό του client την επεξεργασία τους αναλαμβάνει το linux module του πυρήνα του συστήματος. Τα μηνύματα αυτά έχουν αρκετό χώρο έτσι ώστε να μπορούν να επιτρέπουν την δυνατότητα διευθυνσιοδότησης LBA48 του πρωτοκόλλου ATA που είδαμε σε παραπάνω παράγραφο. Αν κάτι μπορεί να μας παραξενέψει είναι ο χώρος που καταλαμβάνει το πεδίο Sector Count, που υποδηλώνει πόσοι sectors βρίσκονται στο μήνυμα, και έχει περιοριστεί στα 8 bits ενώ στο πρότυπο ATA έχει μέγεθος 16 bits. Η αιτία του προβλήματος αυτού είναι οι περιορισμοί που επιβάλλει το Ethernet με το μέγιστο μέγεθος του μηνύματος που για το Ethernent 100 είναι τα 1500 bytes ενώ για το Gigabit Ethernet είναι 9018 bytes. Έτσι βλέπουμε ότι σε κάθε περίπτωση το μέγεθος του πεδίου αυτού είναι ικανό να καλύψει το μέγιστο αριθμό από sectors που μπορούμε να μεταφέρουμε σε κάθε πακέτο.

2.2.2.3 Μηνύματα Ρύθμισης και Ελέγχου

Τα μηνύματα αυτά τα είδαμε έως τώρα να χρησιμοποιούνται κατά τα αρχικά broadcast μηνύματα που στέλνει ο client ή ο server για την εύρεση του δεύτερου. Παράλληλα όμως αυτά τα μηνύματα μεταφέρουν πληροφορίες που είναι αναγκαίες για την αρχικοποίηση τόσο του client όσο και του server. Έτσι ο client μπορεί να αποκτήσει πληροφορίες σχετικά με τον αριθμό των διαθέσιμων buffers που έχει ο server, και να γνωρίζει στην ουσία το μέγιστο αριθμό από μηνύματα που μπορεί να βρίσκονται σε αναμονή στον server. Επίσης μπορεί να αποκτήσει πληροφορίες σχετικά με την συμβολοσειρά ελέγχου του server.

Η συμβολοσειρά ελέγχου περιέχει πληροφορίες ελέγχου που έχει θέσει κάποιος client στον server κατά την εκκίνησή του. Στην ουσία είναι σαν ο client να παίρνει υπό την ιδιοκτησία του τον server, θέτοντας του την τιμή της συμβολοσειράς ελέγχου κατά την εκκίνησή του. Έτσι εμποδίζει κάποιον άλλον client να γράψει στο δίσκο αυτό, επειδή δεν έχει την ίδια συμβολοσειρά ελέγ-

χου. Το χαρακτηριστικό αυτό δεν υλοποιείται από το linux module μέχρι στιγμής.

2.2.2.4 Ακολουθία λειτουργίας

Τελικά η ακολουθία λειτουργίας του πρωτοκόλλου αποδεικνύεται ιδιαίτερα απλή. Αρχικά αποστέλλονται τα αναγκαία μηνύματα ελέγχου για την εύρεση του server και για την ρύθμισή τους. Στην συνέχεια αποστέλλονται εντολές ATA και ξεκινά η ανταλλαγή δεδομένων. Τα χαρακτηριστικά της συσκευής αναλαμβάνει η συσκευή γίνονται γνωστά με την αποστολή της ATA εντολής IDENTIFY DEVICE. Στη συνέχεια ανταλλάσσονται κυρίως εντολές WRITE EXT και READ EXT. Σε περίπτωση διακοπής της λειτουργίας του server ο client ακολουθεί την ίδια τακτική με την εκκίνηση για την εύρεσή του.

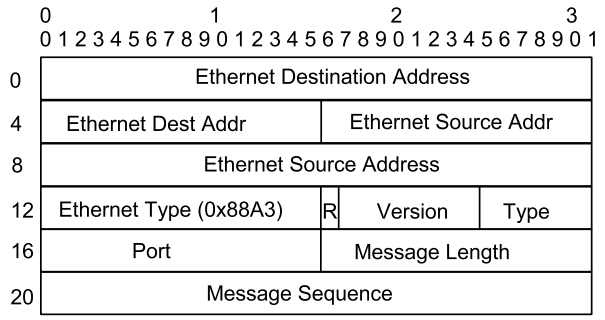
2.3 Το πρωτόκολλο myL3

Μετά την μελέτη του AoE είμαστε σε θέση να αναπτύξουμε ένα παρόμοιο πρωτόκολλο πάνω από το επίπεδο του Ethernet. Σκοπός δεν είναι να υπολοποιήσουμε ξανά υπάρχουσες λύσεις. Το πρόβλημα είναι ότι το AoE δημιουργήθηκε και αναπτύχθηκε υπό την προυπόθεση της χρήσης μιας συγκεκριμένης πλατφόρμας λειτουργίας και υπόκειται στους περιορισμούς που αυτή επιβάλλει. Στόχος μας είναι να αναπτύξουμε ένα παρόμοιο πρωτόκολλο για την διασύνδεση των ελεγκτών ενός συστήματος vRAID με τους σκληρούς δίσκους που βρίσκονται σε κάποιο απομακρυσμένο σύστημα κάνοντας χρήση μόνο του πρωτοκόλλου Ethernet. Θα αντικαταστήσουμε λοιπόν τα επίπεδα IP και TCP που χρησιμοποιεί ήδη η υπάρχουσα μέθοδος επικοινωνίας με το επίπεδο του myL3 που θα αναλάβει να μεταφέρει τα μηνύματα επικοινωνίας του vRAID.

Το πρωτόκολλο αυτό θα πρέπει να εξασφαλίζει τη διατήρηση της σύνδεσης μεταξύ ενός client και server αντιμετωπίζοντας πιθανές απώλειες πακέτων, καθώς και να είναι σε θέση να την αναφέρει. Επίσης θα πρέπει να έχουμε και λειτουργίες ελέγχου ροής των πακέτων (flow control), έτσι ώστε να μην έχουμε άσκοπα χαμένα πακέτα στο σύστημα. Στη λύση μας θεωρούμε ότι οι απώλειες δεν μπορεί να προέλθουν από κάποιο σφάλμα στο επίπεδο δικτύου, αλλά μόνο λόγω συμφόρησης του δικτύου. Για αυτό το λόγο επιβάλλεται η υλοποίηση μηχανισμού ελέγχου ροής που μπορεί να είναι είτε στατικός είτε δυναμικός.

Παρακάτω ακολουθεί η περιγραφή των διαφόρων πακέτων που στέλνει το εν λόγω πρωτόκολλο.

2.3.1 Επικεφαλίδα του Πρωτοκόλλου



Σχήμα 2.7: Επικεφαλίδα του πρωτοκόλλου

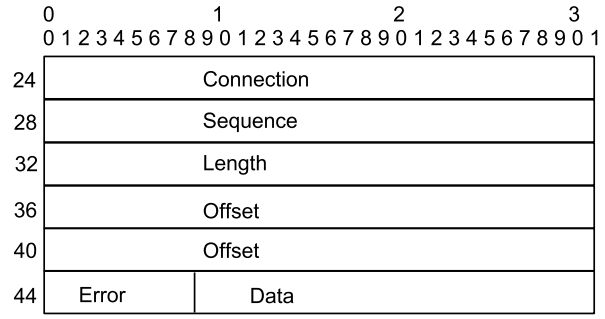
- **Ethernet Destination Address:** Η διεύθυνση Ethernet του παραλήπτη.
- **Ethernet Source Address:** Η διεύθυνση Ethernet του αποστολέα.
- **EthernetType:** Ο κωδικός Ethernet του τύπου του πρωτοκόλλου. Χρησιμοποιείται για να ξεχωρίζει τα διάφορα πακέτα των άλλων πρωτοκόλλων που κάνουν χρήση του Ethernet. Αυθαίρετα επιλέξαμε τον κωδικό 0x88A3
- **R bit:** Αν έχει την τιμή 1 τότε ο πακέτο αυτό είναι απάντηση από κάποιον server. Στην περίπτωση που είναι αίτηση από κάποιον client τότε θα πρέπει να έχει την τιμή 0.
- **Version:** Ο αριθμός έκδοσης του πρωτοκόλλου που υλοποιείται.
- **Type:** Ο τύπος της εντολής vRAID που ακολουθεί. Ανάλογα με την εντολή μπορεί να είναι και διαφορετικός και ο τύπος του μηνύματος που ακολουθεί της επικεφαλίδας
- **Port:** Ο αριθμός διευθυνσιοδότησης του σκληρού δίσκου. Στο όλο δίκτυο Ethernet κάθε σκληρός θα έχει τον δικό του μοναδικό αριθμό.
- **Message Length:** Το συνολικό μέγεθος του μηνύματος.
- **Message Sequence:** Ο αριθμός ακολουθίας του μηνύματος, είναι μοναδικός για κάθε μήνυμα που στέλνει ο πελάτης.

Το πεδίο type μπορεί να λάβει τις τιμές που φαίνονται στον πίνακα. 2.1

Εντολή	Σημασία
CONF	Πρόκειται για μήνυμα ελέγχου
CMD_WRITE	Εντολή εγγραφής
CMD_READ	Εντολή ανάγνωσης
CMD_LOCK	Εντολή κλειδώματος
CMD_UNLOCK	Εντολή ξεκλειδώματος
CMD_RLOCK	Εντολή κλειδώματος και ανάγνωσης
CMD_WUNLOCK	Εντολή εγγραφής και ξεκλειδώματος
CMD_RECSTART	Εντολή έναρξης αποκατάστασης
CMD_RECEND	Εντολή τερματισμού αποκατάστασης
CMD_RECNEXT	Εντολή επόμενων sectors προς αποκατάσταση

Πίνακας 2.1: Η τιμές του πεδίου Type

2.3.2 Πακέτο Μεταφοράς Δεδομένων myL3



Σχήμα 2.8: Πακέτο μεταφοράς δεδομένων του vRAID

Το πακέτο αυτό περιλαμβάνει όλες τις αναγκαίες πληροφορίες που περιέχονται στα πακέτα επικοινωνίας του vRAID, και είναι αναγκαία για την εκτέλεση των διαφόρων εντολών.

- **Connection:** Ο αριθμός σύνδεσης στον controller.
- **Sequence:** Ο αριθμός αλληλουχίας του μηνύματος στο επίπεδο vRAID.
- **Length:** Ο αριθμός των sectors που θα εκτελεστεί η εντολή.
- **Offset:** Η θέση του πρώτου sector απ' όπου θα αρχίσει να εφαρμόζεται η εντολή.

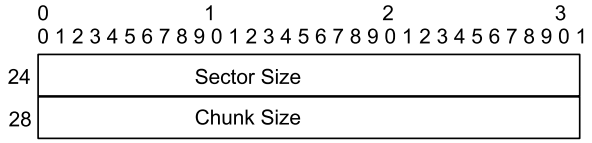
- **Error:** Ο κωδικός του αποτελέσματος της εκτέλεσης της εντολής.
- **Data:** Τα δεδομένα που περιέχονται για την εκτέλεση μιας εντολής

Το πεδίο Error μπορεί να πάρει τις τιμές που φαίνονται στον πίνακα 2.2

Κωδικός σφάλματος
RESP_WRITE_OK
RESP_WRITE_ALL_OK
RESP_WRITE_ERR
RESP_WRITE_ERR_LOCKED
RESP_READ_OK
RESP_READ_ERR
RESP_LOCK_OK
RESP_LOCK_ERR
RESP_UNLOCK_OK
RESP_UNLOCK_ERR
RESP_RLOCK_OK
RESP_RLOCK_ERR
RESP_WUNLOCK_OK
RESP_WUNLOCK_ERR
RESP_UNKNOWN_CMD
RESP_ARRIVED_OK

Πίνακας 2.2: Τιμές που μπορεί να πάρει το πεδίο Error, επιβεβαίωσης ή σφάλματος των αντίστοιχων εντολών.

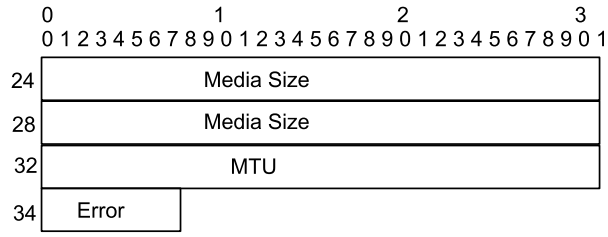
2.3.3 Πακέτο Ελέγχου - Αίτηση



Σχήμα 2.9: Πακέτο ελέγχου - Αίτηση

- **Sector Size:** Το μέγεθος που έχει ο sector στο σύστημα του vRAID. Η τυπική του τιμή είναι 512 bytes.
- **Chunk Size:** Το μέγιστο μέγεθος σε sector που μπορεί να επηρεάσει μια εντολή vRAID.

2.3.4 Πακέτο Ελέγχου - Απάντηση



Σχήμα 2.10: Πακέτο ελέγχου - απάντηση

Κωδικός σφάλματος	Σημασία
CONF_MED_OK	Απουσία σφάλματος
CONF_MED_ERR	Σφάλμα κατά την προσπέλαση του δίσκου.

Πίνακας 2.3: Τιμές που μπορεί να λάβει το πεδίο Error

- **Media Size:** Το μέγεθος του σκληρού δίσκου σε sectors.
- **MTU:** Ο μέγιστος αριθμός byte που μπορεί να μεταφερθεί από κάποιο πακέτο Ethernet. Η τυπική του τιμή είναι 1500 bytes. Στο Gigabit Ethernet μπορεί να φτάσει μέχρι και τα 9000 αν υποστηρίζεται από τις συσκευές δικτύωσης.
- **Error:** Ο κωδικός λάθους σε περίπτωση σφάλματος κατά την επεξεργασία της αίτησης του μηνύματος ελέγχου.

2.3.5 Περιγραφή Λειτουργίας

Η λογική που ακολοθούμε για την υλοποίηση των λειτουργιών του πρωτοκόλλου είναι παρόμοια με αυτή της λειτουργίας της AoE. Έτσι μπορούμε να ξεχωρίσουμε δύο φάσεις λειτουργίας. Την φάση εκκίνησης όπου γίνεται η εύρεση του δίσκου στο δίκτυο με την εγκατάσταση της σύνδεσης μεταξύ του client και του server. Και την φάση της επικοινωνίας όπου ανταλλάσονται μηνύματα του πρωτοκόλλου του vRAID.

2.3.5.1 Φάση αρχικοποίησης της λειτουργίας

Κατά την έναρξή του ένας client το μόνο που είναι σε θέση να γνωρίζει είναι ο αριθμός port του δίσκου που θέλει να συνδεθεί. Για να μπορέσει να εντοπίσει

την διεύθυνση mac του server ξενικά με την αποστολή broadcast πακέτων ελέγχου. Στο μήνυμα αυτό θα απαντήσει μόνο ο server - σκληρός δίσκος που έχει το συγκεκριμένο port. Έτσι είμαστε σε θέση να ανακαλύψουμε την mac διεύθυνση του server , που μπορεί για κάποιο τρόπο να έχει αλλάξει. Στην φάση αυτή ανταλλάσσονται επίσης και πληροφορίες αναγκαίες για την ρύθμιση της λειτουργίας του client και του server, όπως αυτές φαίνονται στην αίτηση και στην απάντηση του μηνύματος ελέγχου.

Επίσης κατά την διαδικασία της αρχικοποίησης γίνεται ο αρχικός υπολογισμός του Round Trip Time (RTT). Για τον υπολογισμό του γίνεται αποστολή δέκα συνολικά μηνυμάτων ελέγχου για να είναι καλύτερη η προσέγγιση που θα κάνουμε στον χρόνο αυτόν. Επίσης κατά την διαδικασία επεξεργασίας των αιτήσεων ελέγχουν στον server γίνεται παράλληλη και μια ανάγνωση από τον σκληρό δίσκο, έτσι ώστε ο RTT χρόνος να προσεγγίζει αυτόν που θα έχουμε κατά την λειτουργία του συστήματος, όπου εκτός από την επεξεργασία των μηνυμάτων θα έχουμε και αναγνώσεις - εγγραφές με τον σκληρό δίσκο, που μπορεί να εισάγουν στην ουσία την καθυστέρηση στο σύστημα.

2.3.5.2 Φάση επικοινωνίας

Σε αυτή την φάση έχουμε την αποστολή μηνυμάτων myL3. Σκοπός του πρωτοκόλλου είναι να εξασφαλίσει την μεταφορά και την επιτυχημένη εκτέλεση των εντολών του controller. Έτσι αφού λάβει μια αίτηση vRAID από το παραπάνω επίπεδο παράγει τα ανάλογα πακέτα myL3. Κάθε πακέτο έχει τον δικό του Sequence Number. Όταν ο server στέλνει την απάντησή του σε κάποια αίτηση πρέπει να αντιγράψει τον Sequence Number της επικεφαλίδας του myL3. Έτσι ο client θα είναι σε θέση να αφαιρέσει από την αντίστοιχη λίστα την αίτηση. Σε περίπτωση που για κάποιο πακέτο δεν λάβουμε απάντηση μέσα σε εύλογο χρονικό διάστημα τότε τα αποστέλλουμε ξανά.

2.3.5.3 Υπολογισμός RTT

Ο αλγόριθμος που χρησιμοποιήσαμε για τον υπολογισμό του μέσου RTT βασίζεται στον αλγόριθμο που προτείνεται στο [16] και χρησιμοποιείται στο TCP.

```
m = m < RTT_MIN ? RTT_MIN : m ;
/* update averate estimator */
m -= (sa >>3);
sa +=m;
/* update deviation estimator */
if (m<0) m=-m;
m -= (sv >> 2);
```

```
sv += m;
rto = (sa >> 3 ) + sv ;
```

Οι μεταβλητές που εμφανίζονται στον παραπάνω αλγόριθμο είναι: m είναι τρέχουσα η μέτρηση τους RTT, sa είναι η μέση τιμή του RTT (a) και sv η απόκλιση του (v). Η νέα μέση τιμή του RTT rto ορίζεται ως $rto = a + 4v$. Η ελάχιστη τιμή του RTT έχει προκύψει από πειραματική παρατήρηση, και έχει τιμή αρκετά μεγάλη ώστε να μην έχουμε άσκοπες αναμεταδόσεις μηνυμάτων αλλά και να μην καθυστερεί πολύ το σύστημα σε περίπτωση που έχουμε κάποια απώλεια μηνύματος. Θα πρέπει να επισημάνουμε ότι η μεγαλύτερη καθυστέρηση προκαλείται από το I/O στο δίσκο και διαδοχικές τιμές του RTT μπορεί να διαφέρουν κατά πολύ. Γι' αυτό και είναι προτιμότερο η ελάχιστη τιμή του RTT να είναι αρκετά μεγάλη, γιατί υπάρχει μεγαλύτερη πιθανότητα ένα μήνυμα να καθυστερήσει κατά το I/O παρά να χαθεί κατά την μετάδοση. Γενικά η απώλεια μηνύματος αντιμετωπίζεται ως σπάνιο γεγονός και η διαδικασία αντιμετώπισής του δεν είναι η βέλτιστη.

2.3.5.4 Έλεγχος ροής

Ο έλεγχος ροής των μηνυμάτων γίνεται αποκλειστικά από τον client. Στην όλη διαδικασία ο server παραμένει παθητικός. Ο client λοιπόν έχει την ευθύνη να περιορίσει τον αριθμό των μηνυμάτων που αποστέλλονται καθώς και να ξαναστείλει μηνύματα για τα οποία μπορεί να μην έχει λάβει απάντηση. Η απώλεια κάποιου μηνύματος μπορεί να συμβεί κατά την αποστολή του μηνύματος από τον client στον server ή να έχουμε απώλεια της απάντησης. Και στις δύο περιπτώσεις ο client θα ξαναστείλει το μήνυμα. Ο μηχανισμός που θα χρησιμοποιηθεί αφήνεται στην εφαρμογή και θα περιγραφεί στην συνέχεια.

Κεφάλαιο 3

Σχεδιασμός Υλοποιήσης Πρωτοκόλλων και Εφαρμογών

3.1 Σχεδιασμός τους AoE

Το AoE πρωτόκολλο αποτελείται από δύο λειτουργικά κομμάτια. Το ένα είναι ο server που είδαμε ότι μπορεί να είναι είτε ένα EtherBlade ή κάποιο πρόγραμμα όπως το δικό μας. Στην πλευρά του client βρίσκεται ο οδηγός συσκευής του Linux (kernel module). Το module αυτό παρέχεται από την ίδια την εταιρία που ανέπτυξε το πρωτόκολλο, βρίσκεται σε αρκετά σταθερή έκδοση και έχει ήδη ενσωματωθεί στον πυρήνα. Στη συνέχεια θα περιγράψουμε τον τρόπο λειτουργίας του διαθέσιμου module και του server που αναπτύξαμε.

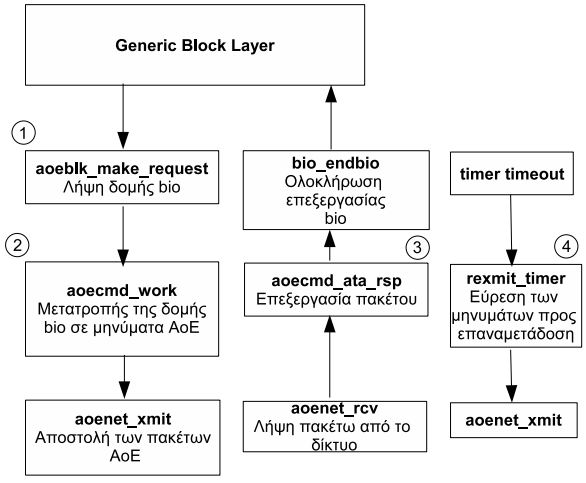
3.1.1 Kernel Module

Η έκδοση του module είναι η 6-29. Όταν εισάγουμε το module στον πυρήνα δημιουργεί κάτω από τον φάκελο /dev/etherd/ τέσσερις συσκευές χαρακτήρων (character devices) με τα ονόματα discover, err, interfaces και revalidate. Επίσης όταν εντοπίζει κάποιον AoE server δημιουργεί την αντίστοιχη συσκευή block, με όνομα της μορφής eX.Z όπου το X αναφέρεται στον αριθμό shelf της συσκευής και το Z στον αριθμό slot. Παράλληλα ο οδηγός δημιουργεί και τις κατάλληλες διεπαφές στο sysfs, όπου κάτω από τον φάκελο /sys/block/etherd!eX.Z περιέχει τις αναγκαίες πληροφορίες για την συσκευή.

Ο σκοπός της ύπαρξης των συσκευών χαρακτήρων έγγειται στην εύκολη επικοινωνία του module με το περιβάλλον, επιτρέποντας την εγγραφή και την ανάγνωση σε αυτές τις συσκευές, χωρίς να είναι αναγκαία η υλοποίηση εντολών ioctl από τη συσκευή block. Έτσι από την συσκευή err έχουμε την δυνατότητα να διαβάσουμε όσα σφάλματα έχουν προκύψει μέχρι εκείνη την

στιγμή κατά τη μετάδοση των διαφόρων πακέτων που πρωτοκόλλου. Γράφοντας στην συσκευή interfaces μπορούμε να θέσουμε σε ποιες δικτυακές διεπαφές (network interfaces) θέλουμε να περιορίσουμε να γίνεται η επικοινωνία με τη χρήση του AoE. Γράφοντας στη συσκευή discover δίνουμε εντολή στον οδηγό να αναζητήσει αν υπάρχουν καινούργιοι AoE servers στο σύστημα. Τέλος με την συσκευή revalidate δίνουμε εντολή ο οδηγός να επανελέγξει το μέγεθος της συσκευής. Για την πραγματοποίηση όλων αυτών των διεργασιών μαζί με το module προσφέρονται και ένα σύνολο από προγράμματα για τη διαχείριση των AoE συσκευών, που στην ουσία χρησιμοποιούν τις παραπάνω συσκευές χαρακτήρων όπως περιγράψαμε.

Είδαμε ότι ο ο οδηγός για την επικοινωνία με το σύστημα δημιουργεί ένα σύνολο από συσκευές χαρακτήρων και block. Εσωτερικά ο οδηγός αποτελείται από τέσσερα δομικά κομμάτια: το σύνολο των συναρτήσεων που είναι αναγκαίες για την συσκευή χαρακτήρων, για την συσκευή block, για την αποστολή και λήψη των πακέτων από δικτυακή διεπαφή, και για την επεξεργασία των εντολών AoE. Η όλη λειτουργία της συσκευής είναι ασύγχρονη. Θα μπορούσαμε να πούμε ότι αποτελείται από ένα κύριο νήμα επεξεργασίας που αναλαμβάνει να επεξεργάζεται της I/O αιτήσεις και να τις μετατρέπει σε πακέτα AoE προς αποστολή. Τα ασύγχρονα γεγονότα που μπορούν να συμβούν μπορεί να είναι η λήψη ενός πακέτου ή να δημιουργηθεί διακοπή από την εκπνοή του χρονομέτρου, που σκοπό έχει την επαναποστολή των πακέτων (για τα οποία δεν έχουμε λάβει απάντηση μέσα σε κάποιο συγκεκριμένο χρονικό διάστημα).



Σχήμα 3.1: Τρόπος λειτουργίας τους AoE module

Η ροή ελέγχου φαίνεται αρκετά αφαιρετικά στο σχήμα: 3.1. Δίνεται η

εντύπωση ότι όπως απεικονίζεται η ροή ελέγχου του προγράμματος, το πρόγραμμα αποτελείται από δύο νήματα. Κάτι τέτοιο όμως δεν ισχύει. Την εκάστοτε στιγμή δυνάμεθα να έχουμε ή την επεξεργασία κάποιας εισερχόμενης δομής bio ή κάποιου εισερχόμενου μηνύματος και αυτό υλοποιείται με την χρήση του κατάλληλου spinlock. Για τις διάφορες λειτουργίες του κώδικα υπάρχουν οι παρακάτω παρατηρήσεις.

1. Για την λήψη των αιτήσεων από το επίπεδο Block χρησιμοποιούμε την συνάρτηση της μορφής make_request και όχι request. Αυτό συμβαίνει γιατί στην συσκευή μας δεν κάνουμε χρήση της δομής request_queue_t αλλά λαμβάνουμε κατευθείαν δομές bio, και δεν παρεμβαίνει κάποιος I/O scheduler για την βελτιστοποίηση των αιτήσεων προς το αποθηκευτικό μέσο. Αυτό συμβαίνει γιατί οι schedulers είναι σχεδιασμένοι να αντιμετωπίζουν φυσικές block συσκευές. Στην περίπτωσή μας όμως δεν υπάρχει κάποιος φυσικός δίσκος που θέλουμε επικοινωνήσουμε, αλλά στην ουσία ένας δικτυακός δίσκος και η επίδοση του συστήματος δεν θα επωφεληθεί από την ύπαρξή του.

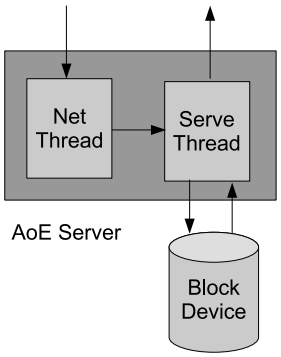
2. Κατά την επεξεργασία των δομών bio, αυτές μπορεί να έχουν μέγεθος αρκετών sector. Όπως όμως έχουμε δει στα πακέτα του AoE μπορούμε να μεταφέρουμε στην ουσία μόνο μέχρι δύο sectors. Μέσα λοιπόν στην συνάρτηση αυτή σπάμε την αίτηση bio σε όσα πακέτα AoE είναι αναγκαία. Για να μπορέσει ο οδηγός να πραγματοποιήσει έλεγχο ροής των δεδομένων, και να μην έχουμε χαμένα πακέτα λόγω υπερχείλισης των buffers του server ή του client υπάρχει ένας περιορισμένος αριθμός από πακέτα που μπορεί να στείλει γενικά. Όταν λάβει κάποια απάντηση τότε δημιουργείται διαθέσιμος χώρος προς αποστολή. Ο αριθμός των διαθέσιμων πακέτων αρχικοποιείται κατά την λήψη του αρχικού μηνύματος ελέγχου. Ανάμεσα στα πεδία ενός τέτοιου πακέτου είναι και ο αριθμός των διαθέσιμων buffers που έχει ο server, και αυτή είναι και η τιμή των διαθέσιμων πακέτων που μπορεί να έχει ο οδηγός. Με αυτόν τον τρόπο δε μπορεί να έχουμε τουλάχιστον χαμένα πακέτα στην πλευρά του server.

3. Κατά την λήψη του πακέτου γίνεται ο υπολογισμός του νέου μέσου RTT. Επίσης αν έχουν ληφθεί όλα τα πακέτα που προέκυψαν από την επεξεργασία κάποιας δομής bio, τότε με την κλήση της bio_endbio, θεωρούμε ότι έχουμε την ολοκλήρωσή της, και το αποτέλεσμα μεταβιβάζεται στο ανώτερο επίπεδο (αυτό του Block Layer).

4. Η συνάρτηση rexmit_timer καλείται περιοδικά με την χρήση ενός χρονομέτρου, το οποίο αρχικοποιείται κάθε φορά στο τέλος της συνάρτησης

αυτής με την τρέχουσα τιμή του RTT. Κύρια λειτουργία της εν λόγω συνάρτησης είναι να βρίσκει ποια μηνύματα έχουμε στην λίστα για τα οποία δεν έχουμε λάβει απάντηση άν και έχει περάσει ο απαιτούμενος χρόνος. Επίσης αν για κάποιο μήνυμα περιμέναμε απάντηση πέραν ενός χρονικού ορίου, σημαίνει ότι έχει χαθεί πλέον η σύνδεση με την συσκευή και ο οδηγός κλείνει στην ουσία την αντίστοιχη block συσκευή που είχε δημιουργηθεί.

3.1.2 AoE Server



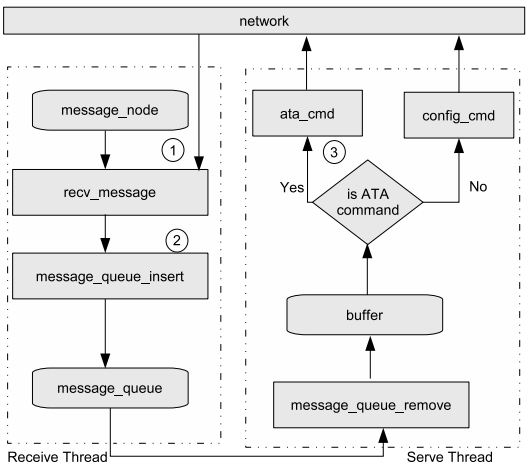
Σχήμα 3.2: Τρόπος λειτουργίας του AoE Server

Όσο περίπλοκος ήταν στο σχεδιασμό και στην υλοποίησή του ο οδηγός του Linux τόσο απλός είναι ο AoE server. Όπως μπορούμε να δούμε στο σχήμα: 3.2 αποτελείται στην ουσία από δύο νήματα. Το ένα έχει τον ρόλο να λαμβάνει τα εισερχόμενα πακέτα και να τα μεταβιβάζει με την σειρά του στο άλλο νήμα που επεξεργάζεται τις εντολές ATA και στέλνει τις απαντήσεις.

1. Λήψη του πακέτου από το δίκτυο. Για να είναι δυνατή η λήψη και η αποστολή Ethernet πακέτων γίνεται χρήση του Packet Interface [1] που προσφέρει την δυνατότητα για την αποστολή και λήψη πακέτων κατευθείαν από το επίπεδο της δικτυακής συσκευής. Η επικοινωνία γίνεται με τη χρήση socket όπως συμβαίνει σε μια τυπική σύνδεση TCP.

2. Με την λήψη κάποιου καινούργιου πακέτου, αυτό εισάγεται στην ουρά για την εξυπηρέτησή.

[1] Με την χρήση της εντολής "man 7 packet" μπορείτε να δείτε τα χαρακτηριστικά αυτού του είδους socket

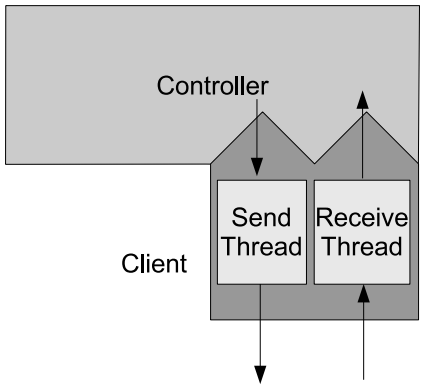


Σχήμα 3.3: Τρόπος λειτουργίας του AoE Server

3. Οι εντολές ATA που μπορεί να διαχειριστεί ο server είναι οι Identify Device, Write, Write EXT, Read και Read EXT. Σε κάθε περίπτωση αυτές είναι οι αναγκαίες εντολές για να είναι δυνατή η επικοινωνία με τον οδηγό του Linux.

3.2 MyL3

3.2.1 MyL3 client



Σχήμα 3.4: Client του πρωτοκόλλου myL3

Όπως Βλέπουμε στο σχήμα: 3.4, ο client του myL3 είναι μέρος του

vRAID Controller. Όπως και στην περίπτωση του AoE client, αυτός είναι που αναλαμβάνει την όλη ευθύνη της επικοινωνίας με τον server, και παραμένει ένας παθητικός διεκπεραιωτής εντολών εγγραφής και ανάγνωσης.Αποτελείται από δύο νήματα εκ των οποίων το ένα αναλαμβάνει την αποστολή των μηνυμάτων και το άλλο τη λήψη τους.

Το νήμα της αποστολής λαμβάνει από την ουρά εισερχομένων αιτήσεων τις εντολές από τον controller. Αυτές μπορεί να είναι εγγραφές, αναγνώσεις ή οποιεσδήποτε άλλες εντολές προς τον απομακρυσμένο δίσκο. Επειδή το μέγιστο μήκος του μηνύματος περιορίζεται από το MTU του Ethernet, το νήμα αποστολής αναλαμβάνει να παράγει τα αντίστοιχα μηνύματα myL3 που με τη σειρά τους εισάγονται στην ουρά με τα έτοιμα μηνύματα.

Ο μηχανισμός του flow-control αναλαμβάνει να στείλει τα καινούργια έτοιμα μηνύματα με την προϋπόθεση ότι το παράθυρο είναι άδειο ως στη μέση. Με την αποστολή του μήνυματος σημαδεύεται μ' ένα timestamp και εισέρχεται στην λίστα των απεσταλμένων μηνυμάτων.

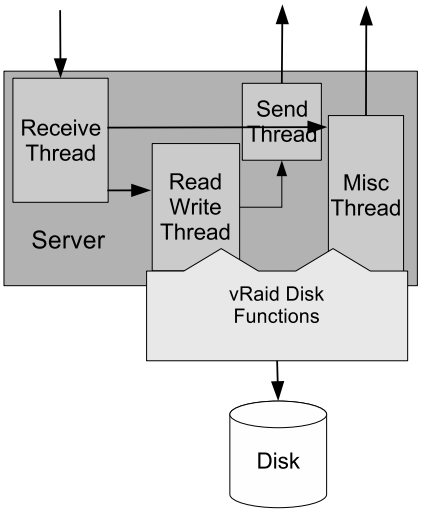
Το νήμα της λήψης διαβάζει το εισερχόμενο μήνυμα, και το αναζητά μέσα στην λίστα με τα απεσταλμένα μηνύματα από όπου και το αφαιρεί. Ανάλογα με το είδος του εισερχόμενου μηνύματος, αν είναι δηλαδή μια επιβεβαίωση ή αν είναι κομμάτι κάποιας μεγαλύτερης αίτησης, το μεταβιβάσει στην ουρά απαντήσεων ή το αποθηκεύει προσωρινά σε μια λίστα μέχρι να φτάσουν και τα εναπομένοντα κομμάτια. Έτσι θα μπορέσει να στείλουμε ολοκληρωμένη την απάντηση στον controller. Περιοδικά το νήμα λήψης ελέγχει αν κάποιο από τα σταλμένα μηνύματα έχει υπερβεί τον χρόνο αναμονής, δηλαδή αν δεν έχουμε λάβει απάντηση μέσα σε χρονικό διάστημα RTT οπότε αποστέλλεται εκ νέου στον server.

3.2.2 MyL3 server

Ο server με τη σειρά του είναι αρκετά απλός (σχήμα 3.5). Αποτελείται από τέσσερα νήματα, ένα για την λήψη των μηνυμάτων, ένα για την πραγματοποίηση εγγραφών και αναγνώσεων στον δίσκο, ένα για την αποστολή των απαντήσεων από τον δίσκο και ένα για την υλοποίηση όλων των υπολοίπων εντολών.

3.3 Υλοποίηση με χρήση Myrinet

Όπως θα δούμε και στην παράγραφο: 5.2 που περιγράφει το GM API η ανάπτυξη εφαρμογών με τη χρήση αυτής της βιβλιοθήκης διαφέρει πάρα πολύ από τον τρόπο προγραμματισμού που έχουμε συνηθίσει με τη χρήση των Unix Sockets. Πρόκειται για ένα API που βασίζεται στη μετάδοση μηνυμάτων και

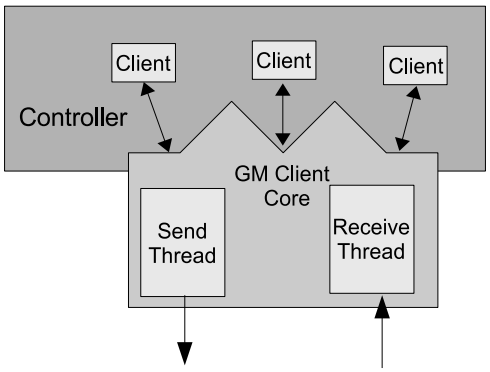


Σχήμα 3.5: Server του πρωτοκόλλου myL3

προσπαθεί να εκμεταλλευτεί τις δυνατότητες της κάρτας Myrinet στο έπακρο. Έτσι προσφέρει περιορισμένες διεπαφές για την αποστολή και τη λήψη μηνυμάτων, περιορίζοντας σημαντικά τις προγραμματιστικές μονάδες που μπορούν να επικοινωνήσουν απευθείας με την κάρτα και επιβάλλοντας τη δημιουργία αρχιτεκτονικής δύο επιπέδων όταν θέλουμε να έχουμε περισσότερες από μια "συνδέσεις" με απομακρυσμένους υπολογιστές.

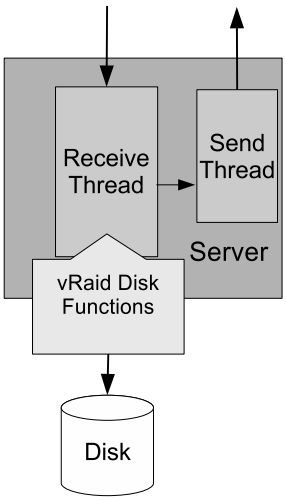
3.3.1 GM client

Στο σχήμα: 3.6 βλέπουμε τη διάταξη του client για το πρωτόκολλα GM. Μπορούμε να παρατηρήσουμε την αρχιτεκτονική δύο επιπέδων που περιγράψαμε παραπάνω. Έχουμε δηλαδή την ύπαρξη ενός πυρήνα που αναλαμβάνει την αποστολή και τη λήψη των μηνυμάτων με τη χρήση δύο αντίστοιχων νημάτων. Οι επιμέρους clients πρέπει να συνδέονται με τον πυρήνα και να αποστέλλουν μέσω αυτού τα μηνύματά τους.



Σχήμα 3.6: Ο τρόπος λειτουργίας τους Client με τη χρήση του πρωτοκόλλου GM

3.3.2 GM server



Σχήμα 3.7: Ο Server για το πρωτόκολλο GM

Ο server ακολουθεί και εδώ την απλοϊκή υλοποίηση έχοντας αυτή τη φορά μόνο 2 νήματα για λόγους που θα εξηγήσουμε στην περιγραφή της υλοποίησης. Το ένα νήμα χρησιμοποιείται για τη λήψη και των εισερχομένων μηνυμάτων και την άμμεση εκτέλεση της εντολής, ενώ το δεύτερο για την αποστολή των απαντήσεων.

Κεφάλαιο 4

Υλοποίηση vRAID

Παρακάτω περιγράφεται αναλυτικά η λειτουργία της παρούσας έκδοσης του vRAID. Η περιγραφή έχει χωριστεί στα διάφορα δομικά κομμάτια του vRAID όπως αυτά θα αναλυθούν στις παρακάτω ενότητες.

4.1 Διεπαφές Επικοινωνίας Χρήστη

Το σύστημα vRAID ως ένα τυπικό σύστημα SAN θα πρέπει να προσφέρει στους clients του διεπαφές επικοινωνίας κατάλληλες που θα επιτρέπουν την επικοινωνία σε επίπεδο block. Όπως περιγράψαμε και στο αρχικό κεφάλαιο το vRAID προσπαθεί να συμπεριλάβει στις μεθόδους επικοινωνίας του και να υποστηρίξει τεχνολογίες που να είναι χαμηλού κόστους και ευρέως διαδεδομένες. Έτσι στον αρχικό του σχεδιασμό για την επικοινωνία με τους clients γίνεται χρήση της τεχνολογίας Ethernet και των πρωτοκόλλων TCP/IP. Ως λύσεις για την επικοινωνία σε επίπεδο block μεταξύ του controller και clients επιλέχθηκαν υφιστάμενες λύσεις που παρέχονται στα συστήματα Linux και FreeBSD για τις οποίες αναπτύχθηκαν κατάλληλοι server. Οι λύσεις που έχουν υλοποιηθεί μέχρι στιγμής είναι αυτών του Network Block Device (NBD) του Linux του Ggate στο FreeBSD και του πρωτοκόλλου AoE της εταιρίας Coraid που προσφέρει οδηγούς και στις δύο εκδόσεις του Unix. Οι δύο πρώτες λύσεις περιγράφονται παρακάτω ενώ η υλοποίηση του AoE είναι μέρος αυτης της διπλωματικής εργασιάς και περιγράφεται στη παράγραφο: 5.3. Κοινό χαρακτηριστικό και των τριών υλοποιήσεων είναι ότι η επικοινωνία είναι σύγχρονη.

4.1.1 NBD

Ο οδηγός NBD για το Linux είναι μια από τις πρώτες προσπάθειες που έγιναν για να δημιουργηθούν δικτυακές συσκευές αποθήκευσης που να λει-

τουργούν σε επίπεδο block. Εμφανίστηκε αρχικά το 1997 οπότε και έγινε module του πυρήνα του Linux. Από τότε έχουν προκύψει και βελτιώσεις του όπως το ENBD. Κύρια δυνατότητά του είναι ότι επιτρέπει την εξαγωγή μιας οποιασδήποτε συσκευής ή αρχείου ως εικονικό δίσκο στο δίκτυο, τον οποίο μπορεί να προσπελάσει οποιοσδήποτε άλλος υπολογιστής του δικτύου ως μια τυπική συσκευή block του Linux.

Δομικά αποτελείται από το module του πυρήνα και δύο βοηθητικές εφαρμογές, τους client και server daemons, που επιτρέπουν την μεταφορά των εντολών έναρξης και ρύθμισης του module από το περιβάλλον του χρήστη σε αυτό του πυρήνα. Το module εσωτερικά αποτελείται και αυτό από δύο νήματα αντίστοιχα των εφαρμογών server και client. Κάποια αίτηση εγγραφής ή ανάγνωσης στην block συσκευή που έχουμε δημιουργήσει στο Linux, θα διασχίσει τα διάφορα επίπεδα του πυρήνα και θα φτάσει στο kernel νήμα, το οποίο και θα αναλάβει να τη μετατρέψει σε μια αντίστοιχη αίτηση του πρωτοκόλλου NBD και να τη μεταδώσει στο δίκτυο. Η αίτηση μετά την αποστολή της εισάγεται σε ουρά αναμονής της απάντησης από το άλλο άκρο της επικοινωνίας. Το client νήμα με την σειρά του μένει μπλοκαρισμένο περιμένοντας κάποιο εισερχόμενο μήνυμα. Με την άφιξη της απάντησης αφαιρεί την αντίστοιχη αίτηση από την ουρά αναμονής και μεταφέρει το αποτέλεσμα της απάντησης μέσω των υφισταμένων επιπέδων της αρχιτεκτονικής του Linux στην εφαρμογή του χρήστη που είχε κάνει την ανάλογη αίτηση.

Η μορφή των αιτήσεων και των απαντήσεων που στέλνει το πρωτόκολλο φαίνονται παρακάτω:

```
struct nbd_request {
        uint32_t magic;
        uint32_t type;  /* == READ || == WRITE  */
        char handle[8];
        uint64_t from;
        uint32_t len;
};
struct nbd_reply {
        uint32_t magic;
        uint32_t error;          /* 0 = ok, else error   */
        char handle[8];          /* handle you got from request  */
};
```

Η πληροφορία που μεταφέρουν είναι το είδος της εντολής (εγγραφή ή ανάγνωση), η θέση στο δίσκο που θα αρχίσει τα εκτελείται η εντολή καθώς και το μήκος της. Αν η εκτέλεση της εντολής επιβάλλει να συνοδεύεται και από δεδομένα, τότε αυτά στέλνονται ακριβώς μετά την αποστολή της επικεφαλίδας που είδαμε παραπάνω. Τη σειριακή μετάδοση των δεδομένων και την αποτροπή οποιασδήποτε αλλαγής στη σειρά αποστολής των επικεφαλίδων και των δεδομένων τους μας τα εξασφαλίζει το TPC/IP.

Ο server που έχει γραφτεί ως μέρος του vRAID δεν ακολουθεί την ίδια φιλοσοφία με το module του πυρήνα και αποτελείται από πολλαπλά νήματα. Αρχικά δημιουργείται ένα και βασικό νήμα που έχει σα σκοπό απλώς να αναμένει για καινούργιες TCP συνδέσεις. Με την άφιξη κάποιας καινούργιας σύνδεσης, άρα και την άφιξη ενός καινούργιου client, δημιουργείται ένα καινούργιο νήμα που έχει ως σκοπό την εξυπηρέτηση των αιτήσεων από τον συγκεκριμένο client. Κατά την άφιξη κάποιας καινούργιας αίτησης, αναγνωρίζεται ο τύπος της (αν πρόκειται για εγγραφή ή ανάγνωση) και στη συνέχεια καλείται η αντίστοιχη συνάρτηση που προσφέρει η διεπαφή του vRAID (4.2.3).

4.1.2 Ggate

Το Ggate είναι μια αντίστοιχη προσπάθεια με το NBD του Linux, μόνο που είναι προσαρμοσμένο στα χαρακτηριστικά λειτουργίας του πυρήνα του FreeBSD. Και αυτό με την σειρά του επιτρέπει τοπικά αρχεία ή δίσκοι να προσπελαύνονται από απομακρυσμένα συστήματα, με την ιδιαιτερότητα ότι τα συστήματα αντιμετωπίζονται ως συσκευές block. Η περιγραφή της λειτουργίας του οδηγού του FreeBSD είναι ποιο περίπλοκή από την αντίστοιχη του NBD και ξεφεύγει από τους σκοπούς του παρόντως κειμένου.

Ο server που γράφτηκε για το vRAID ακολουθεί την ίδια λογική με τον αντίστοιχο για το NBD. Αρχικά περιμένει για κάποια εισερχόμενη σύνδεση και όταν αυτή πραγματοποιηθεί τότε εκτελείται η συνάρτηση handshake(), που θα αναλάβει να πραγματοποιήσει τη χειραψία του πρωτοκόλλου Ggate. Με την ολοκλήρωση της σύνδεσης θα δημιουργηθεί μια νέα διεργασία με την κλήση της συνάρτησης fork που θα περιέχει τρία νήματα. Ένα για την λήψη των μηνυμάτων, ένα άλλο για την αποστολή και ένα τρίτο για την πραγματοποίηση της επικοινωνίας με το vRAID ελεγκτή. Να σημειώσουμε εδώ ότι ο Ggate χρησιμοποιεί δύο sockets για την επικοινωνία του: ένα για την λήψη και ένα για την αποστολή. Με την λήψη του καινούργιου μηνύματος, αυτό εισάγεται στην ουρά εισόδου για την εξυπηρέτησή του από τον ελεγκτή. Το νήμα επεξεργασίας λαμβάνει την καινούργια εντολή από το την ουρά εισόδου, την πραγματοποιεί και εισάγει το μήνυμα απάντησης στη ουρά εξόδου. Τέλος το νήμα αποστολής, αφαιρεί τις απαντήσεις από το νήμα εξόδου, και τις αποστέλλει στον client.

4.2 Κεντρικός Ελεγκτής

4.2.1 Αρχικοποίηση

Πριν από την έναρξη της λειτουργίας του ελεγκτή το σύστημα θα πρέπει να αρχικοποιηθεί. Η κεντρική struct του συστήματος είναι η παρακάτω και η περιγραφή των πεδίων της ακολουθεί.

```
struct vraid_info {
        struct {
                in_addr_t       host;
                unsigned        port;
        } server, sync;
        uint64_t                mediasize;
        uint32_t                sectorsize;
        uint32_t                chunksize;
        uint8_t                 raidlevel;
        uint8_t                 raidstatus;
        uint64_t                cachesize;
        uint8_t                 cachepolicy;
        struct vraid_device     *device;
        uint8_t                 devicenum;
        struct vraid_device     *spare;
        uint8_t                 sparenum;
};
```

Πεδίο	Σημασία
server	Η διεύθυνση και η θήρα που λαμβάνει ο server τις αιτήσεις
sync	Η διεύθυνση και η θήρα συγχρονισμού του συστήματος
mediasize	Το μέγεθος του εικονικού δίσκου
sectorsize	Το μέγεθος sector που χρησιμοποιεί το σύστημα
chunksize	Το μέγιστο μέγεθος μιας εντολής vRAID προς τους δίσκους
raidlevel	Το επίπεδο RAID του υλοποιείται
raidstatus	Η κατάσταση του RAID
cachesize	Το μέγεθος της μνήμης cache του συστήματος
cachepolicy	Η πολιτική αντικατάστασης των περιεχομένων στην cache
device	Λίστα από διαθέσιμους δίσκους
devicenum	Αριθμος δίσκων
spare	Λίστα από μη χρησιμοποιούμενους δίσκους
sparenum	Αριθμός μη χρησιμοποιούμενων δίσκων

4.2.2 Διαχείριση Συνδέσεων

Ανωτέρω (4.1) περιγράψαμε τους τρόπους με του οποίους μπορεί να συνδεθεί ένας πελάτης στον controller. Κατά την σύνδεσή του πελάτη δημιουργείται στη ουσία μια νέα καταχώρησή του μέσα στον controller όπως φαίνεται στην δομή που ακολουθεί:

```
struct vraid_connection {
        uint32_t                        connection;
        struct vraid_queue              queue;
```

```
        LIST_ENTRY(vraid_connection)     next;
};

uint8_t vraid_connection_open(uint32_t *connection);
void vraid_connection_close(uint32_t connection);
void vraid_connection_enqueue(uint32_t connection, struct vraid_queue_item
*item);
struct vraid_queue_item *vraid_connection_dequeue(uint32_t connection);
```

Ο αριθμός connection είναι ένα μοναδικός αριθμός ο οποίος ανατίθεται σε κάθε σύνδεση έτσι ώστε το σύστημα να μπορεί να διακρίνει τους πελάτες που είναι συνδεδεμένοι σε αυτό. Θα δούμε παρακάτω ότι ο αριθμός connection θα εμφανίζεται σε όλα τα μηνύματα προς τα απομακρυσμένα αποθηκευτικά συστήματα, με σκοπό να είναι διακριτές οι αιτήσεις μεταξύ τους, και να εισάγονται οι απαντήσεις στην ανάλογη ουρά queue που διαθέτει η κάθε σύνδεση.

Οι συναρτήσεις που ακολουθούν την δομή επιτρέπουν την δημιουργία της (vraid_connection_open()), την διαγραφή της από το σύστημα (vraid_connection_close()), την εισαγωγή μιας καινούργιας απάντησης από μια απομακρυσμένη αποθηκευτική συσκευή στην ουρά queue (vraid_connection_enqueue()) καθώς και την αφαίρεση της τελευταίας απάντησης από την ουρά (vraid_connection_dequeue()).

4.2.3 Εντολές I/O

```
extern uint8_t (*vraid_read)(uint32_t connection, void *buf, uint32_t count,
uint64_t offset);
extern uint8_t (*vraid_write)(uint32_t connection, void *buf, uint32_t count,
uint64_t offset);
```

Οι τυπικοί server των λύσεων που περιγράψαμε στη παράγραφο 4.1 θα περιμέναν σε κάποιο σημείο της εκτέλεσής τους να πραγματοποιήσουν κάποια κλήση συνάρτησης I/O προς το αρχείο ή το δίσκο που διαχειρίζονταν. Αντ' αυτού μπορούν να καλέσουν τις παραπάνω συναρτήσεις για να μπορέσουν να διαβάσουν ή να γράψουν κάποιον αριθμό από blocks στο σύστημα. Η λειτουργία των συναρτήσεων είναι blocking, δηλαδή αναγκάζουν το νήμα που τις κάλεσε να "περιμένει" την ολοκλήρωσή τους μέχρι να μπορέσει να συνεχίσει την λειτουργία του.

Στην ουσία βέβαια δεν υπάρχουν αυτές καθ' εαυτές οι υλοποιήσεις των παραπάνω συναρτήσεων. Πρόκειται για δείκτες σε συναρτήσεις που αντιστοιχούν σε συναρτήσεις που υλοποιούνται, είτε από το υποσύστημα του RAID, είτε από το υποσύστημα της Cache και υλοποιούν τις παρακάτω λειτουργίες. Κατά την αρχικοποίησή του ο controller αναθέτει τις κατάλληλες τιμές στους

παραπάνω δείκτες συναρτήσεων ανάλογα με τις ρυθμίσεις που έχουμε δώσει. Το πλεονέκτημα είναι ότι το υποσύστημα της επικοινωνίας με τους clients καλεί μια συγκεκριμένη συνάρτηση, χωρίς να χρειάζεται να ανησυχεί για ποια μπορεί να είναι η λειτουργία που πραγματοποιείται πραγματικά.

4.2.4 Υλοποίηση RAID

```
void vraid_raid_set_level(void);
extern void (*vraid_raid_check)(void);
extern void (*vraid_raid_init)(void);
extern uint8_t (*vraid_raid_read)(uint32_t connection, void *buf, uint32_t
count, uint64_t offset);
extern uint8_t (*vraid_raid_write)(uint32_t connection, void *buf, uint32_t
count, uint64_t offset);
extern void (*vraid_raid_degrade)(uint32_t connection);
extern void (*vraid_raid_recover)(uint32_t connection);
```

Παραπάνω αναφέρεται η διεπαφή που θα πρέπει να υλοποιούν τα διάφορα διαθέσιμα υποσυστήματα RAID. Όμοια με πριν κατά την αρχικοποίηση του συστήματος στις παραπάνω συναρτήσεις θα ανατεθούν οι τιμές που θα αντιστοιχούν στη συστοιχία RAID που υλοποιεί ο controller. Οι συναρτήσεις vraid_raid_read και vraid_raid_write έχουν ως λειτουργία να κατανείμουν ή να ανακτήσουν τα δεδομένα από τις απομακρυσμένες συσκευές αποθήκευσης. Ο τρόπος που μπορεί να κατανείμουν τα δεδομένα εξαρτάται από το τύπο RAID που υλοποιείται κάθε φορά. Η συνάρτηση vraid_raid_set_level() χρησιμοποιείται κατά την αρχικοποίηση του συστήματος και θέτει τον τύπο του RAID που θα υλοποιήσει το σύστημα. Η συνάρτηση vraid_raid_check() χρησιμοποιείται για να ελέγξει αν οι παράμετροι αρχικοποίησης του συστήματος συμφωνούν με τις απαιτήσεις της συστοιχίας RAID. Η vraid_raid_init() αρχικοποιεί το υποσύστημα RAID. Η vraid_raid_degrade() έχει ως σκοπό να μπορέσει να αντικαταστήσει μια απομακρυσμένη αποθηκευτική συσκευή που για κάποιο λόγο δεν ανταποκρίνεται, με μια από τις διαθέσιμες εφεδρικές και να αποκαταστήσει το σύστημα. Τέλος η vraid_raid_recover() αναλαμβάνει να επαναφέρει τα δεδομένα που καταστράφηκαν με βάση τον αλγόριθμο αποκατάστασης που επιβάλλει ο τύπος του RAID.

4.2.5 Υλοποίηση Cache

```
void vraid_cache_init(void);
uint8_t vraid_cache_read(uint32_t connection, void *buf, uint32_t count,
uint64_t sector);
uint8_t vraid_cache_write(uint32_t connection, void *buf, uint32_t count,
uint64_t sector);
```

Η cache του συστήματος προσφέρει τις ίδιες συναρτήσεις επικοινωνίας με αυτή του υποσυστήματος RAID. Έτσι μπορεί να λειτουργήσει διαφανώς σε σχέση με τα υψηλότερα επίπεδα του VRAID, και να αποθηκεύει τοπικά συχνά χρησιμοποιούμενα δεδομένα. Την παρούσα στιγμή βρίσκεται ακόμα σε φάση ανάπτυξης. Τελικός σκοπός είναι να δημιουργηθούν πλήρως συγχρονισμένες caches, σαν να πρόκειται για RAID 1. Έτσι όταν εγκαταστήσουμε περισσότερους από έναν controllers, και ένας από αυτούς αποτύχει τότε να μπορούν οι υπόλοιποι να τον αντικαταστήσουν.

4.3 Διεπαφή Επικοινωνίας με Απομακρυσμένες Συσκευές Αποθήκευσης

```
struct vraid_device {
        in_addr_t               host;
        unsigned                port;
        uint64_t                mediasize;
        uint8_t                 status;
        struct vraid_queue      queue;
        struct vraid_queue      pending;
};
```

Πεδίο	Σημασία
host	Διεύθυνση που βρίσκεται ο απομακρυσμένος δίσκος
port	Socket port που γίνεται η σύνδεση
mediasize	Το μέγεθος του απομακρυσμένου δίσκου
status	Κατάσταση του απομακρυσμένου δίσκου
queue	Ουρά εισερχόμενων αιτήσεων
pending	Ουρά αιτήσεων υπό αναμονή απάντησης

Το vRAID δεν απαιτεί την υλοποίηση κάποιας συγκεκριμένης διεπαφής για την επικοινωνία με τα απομακρυσμένα αποθηκευτικά υποσυστήματα. Ο προγραμματιστής θα πρέπει να υλοποιήσει μόνο μια συνάρτηση του τύπου device_connect, η οποία θα έχει ως μια από τις παραμέτρους της τη δομή vraid_device ώστε να μπορεί να λαμβάνει τις εισερχόμενες αιτήσεις από τα υψηλότερα επίπεδα του controller, και να μπορεί να μεταβιβάζει και τις απαντήσεις που λαμβάνει. Αυτή η στρατηγική είναι καλύτερη, γιατί η δημιουργία μιας διεπαφής του τύπου 4.2.3 θα απαιτούσε την ενδιάμεση αποθήκευση των μηνυμάτων, γεγονός που θα εισήγαγε επιπλέον φόρτο στο σύστημα. Η υφιστάμενη λύση έχει υλοποιηθεί με την χρήση TCP sockets. Για τις ανάγκες

αυτής διπλωματικής εργασίας έχουμε αναπτύξει παρόμοιες λύσεις με την χρήση raw πακέτων Ethernet(5.4) και στο σύστημα διασύνδεσης Myrinet με την χρήση του GM API(5.5).

Η υπάρχουσα υλοποίηση σε TCP/IP αποτελείται από δύο νήματα, ένα για την αποστολή των αιτήσεων στην απομακρυσμένη συσκευή και ένα για την λήψη των απαντήσεων, που έχουν ανοίξει και αντίστοιχα sockets για την πραγματοποίηση της επικοινωνίας. Η δομή των επικεφαλίδων του κάθε μηνύματος φαίνεται στην παράγραφο 4.3.1. Αν η εντολή συνοδεύεται και από δεδομένα τότε αυτά θα αποσταλούν αμέσως μετά την αποστολή της επικεφαλίδας. Οι μηχανισμοί των sockets μας εξασφαλίζουν την σειριακή μετάδοση των δεδομένων και αποκλείουν την πιθανότητα να έχουμε αντιμετάθεση της σειράς άφιξης των δεδομένων. Έτσι έχουμε την αφαίρεση του πρώτου κατά σειρά μηνύματος προς αποστολή από την ουρά queue, την αποστολή του, και την τοποθέτησή του στην ουρά pending προς αναμονή της απάντησης. Το νήμα λήψης με την σειρά του, όταν ολοκληρώσει την λήψη μιας απάντησης, αφαιρεί την αντίστοιχη αίτηση από την ουρά pending, και προωθεί την απάντηση στο αντίστοιχο connection, καλώντας την συνάρτηση vraid_connection().

Ο server που βρίσκεται στην απομακρυσμένη συσκευή αποθήκευσης, λειτουργεί με την ίδια περίπου λογική, όπως και ο client στον controller. Για κάθε καινούργιο controller που συνδέεται στην συσκευή, δημιουργεί τρία νήματα. Ένα για την λήψη των εισερχόμενων μηνυμάτων που τα εισάγει σε μια ουρά εισόδου. Ένα για την πραγματοποίηση των εντολών I/O και των λοιπών εντολών που μπορεί να σχετίζονται με την διαχείριση του κλείδωματος, όπου με την ολοκλήρωση της εντολής εισάγει την απάντηση στην ουρά εξόδου. Και τέλος ένα νήμα που λαμβάνει τις απαντήσεις από την ουρά εξόδου και τις αποστέλλει στον controller. Όπως και ο client, χρησιμοποιεί και αυτός δύο sockets ένα για την λήψη και ένα για την αποστολή των μηνυμάτων.

4.3.1 Αιτήσεις προς Συσκευές Αποθήκευσης

```
struct vraid_request {
        uint32_t connection;
        uint32_t sequence;
        uint8_t  command;
        uint64_t offset;
        uint32_t length;
};
struct vraid_reply {
        uint32_t connection;
        uint32_t sequence;
        uint32_t length;
        uint8_t  error;
};
```

Η μορφή των επικεφαλίδων των αιτήσεων και των απαντήσεων, καθώς και οι διαθέσιμες εντολές προς τα απομακρυσμένα αποθηκευτικά για την υλοποίηση του συστήματος επικοινωνίας σε TCP/IP φαίνονται παραπάνω. Οι επικεφαλίδες περιέχουν την ελάχιστη απαιτούμενη πληροφορία που χρειάζεται για να μπορέσουμε να αλληλεπιδράσουμε με κάποιον απομακρυσμένο δίσκο. Καθορίζεται η θέση από την αρχή και ο αριθμός σε sectors στο δίσκο θα εφαρμοστεί η κάθε εντολή. Μπορούμε να δούμε επιπλέον ,ότι περιέχεται το πεδίο connection, που μας επιτρέπει να δρομολογήσουμε την απάντηση πίσω στον πελάτη, που ζήτησε την αντίστοιχη εντολή I/O. Επίσης περιέχεται και το πεδίο sequence, που είναι ο αριθμός ακολουθίας του πακέτου, και μας επιτρέπει να ξεχωρίσουμε τις διάφορες αιτήσεις μεταξύ τους.

Εντολή	Σημασία
VRAID_COMMAND_CLOSE	Κλείσιμο της σύνδεσης.
VRAID_COMMAND_READ	Εντολή ανάγνωσης.
VRAID_COMMAND_WRITE	Εντολή εγγραφής.
VRAID_COMMAND_LOCK	Εντολή κλειδώματος sectors.
VRAID_COMMAND_UNLOCK	Εντολή ξεκλειδώματος sectors.
VRAID_COMMAND_RLOCK	Εντολή κλειδώματος και ανάγνωσης.
VRAID_COMMAND_WUNLOCK	Εντολή εγγραφής και ξεκλειδώματος.
VRAID_COMMAND_RECSTART	Εντολή έναρξης αποκατάστασης.
VRAID_COMMAND_RECEND	Εντολή τερματισμού αποκατάστασης.
VRAID_COMMAND_RECNEXT	Εύρεση επομένων sectors προς αποκατάσταση.

Πίνακας 4.1: Οι εντολές που θα πρέπει να υποστηρίζει η απομακρυσμένη αποθηκευτική συσκευή.

Οι διαθέσιμες εντολές φαίνονται στον παραπάνω πίνακα. Όσες αναφέρονται σε sectors εφαρμόζονται σε απόσταση offset από την αρχή του δίσκου και για εύρος length.

4.4 Μηχανισμός Διαχείρισης Κλειδωμάτων

Ο μηχανισμός των κλειδωμάτων υλοποιείται τοπικά σε κάθε απομακρυσμένη αποθηκευτική μονάδα. Υλοποιείται υπό την μορφή ενός bitmap, ενός πίνακα δηλαδή από bits που το καθένα αντιστοιχεί και σ' ένα sector του δίσκου. Αν το bit έχει τιμή 1 τότε σημαίνει ότι ο αντίστοιχος sector είναι κλειδωμένος. Αντίστοιχα αν έχει τιμή 0 σημαίνει ότι δεν είναι κλειδωμένος.

Όπως είδαμε ο μηχανισμός κλειδωμάτων διαθέτει μια μεγάλη πληθώρα από εντολές, οι οποίες τον διαχειρίζονται. Σκοπός του είναι να προσφέρει ασφαλή διαχείριση των δεδομένων, αποτρέποντας ταυτόχρονες εγγραφές, που μπορεί να τα αλλοιώσουν.

Η διαχείριση των κλειδωμάτων γίνεται από τον controller. Ο αλγόριθμος που ακολουθείται είναι αυτός του two-phase locking. Είναι δηλαδή υποχρεωμένος ο controller να πάρει όλα τα κλειδιά, απ' όλους τους δίσκους με τους οποίους θέλει να αλληλεπιδράσει, πριν προχωρήσει στην εγγραφή, ή την ανάγνωση.

Κεφάλαιο 5

Υλοποίηση Εφαρμογών

5.1 Επικοινωνία Ethernet

Η λήψη και η αποστολή απευθείας πακέτων Ethernet δεν είναι συνηθισμένη για μια εφαρμογή, αφού το λογικό είναι να κάνει χρήση των υφιστάμενων πρωτοκόλλων TCP/IP που επιτρέπουν την εύκολη επικοινωνία. Ποιες εφαρμογές εκτός από την δική μας μπορεί λοιπόν να χρειάζονται να λαμβάνουν όλα τα πακέτα από την πηγή τους; Προφανώς όσες εφαρμογές έχουν ως σκοπό τον έλεγχο της κίνησης από κάποια δικτυακή διεπαφή (interface) και την επισκόπηση των πακέτων που περνούν από αυτή. Τέτοιες εφαρμογές που μας έρχονται στο μυαλό είναι το iptraf, tcpdump, ethereal, προγράμματα που χρησιμοποιούμε για να ελέγχουμε τα εξερχόμενα και τα εισερχόμενα πακέτα. Κοινό στοιχείο όλων αυτών των εφαρμογών είναι ότι, για να τρέξουν χρειάζονται αυξημένα δικαιώματα, κοινώς μόνο ο διαχειριστής του συστήματος μπορεί να τα εκτελέσει. Η εξήγηση είναι απλή: δε δύναται ο απλός χρήστης να έχει το δικαίωμα να μπορεί με ευκολία για διαβάσει όλα τα δεδομένα που μετακινούνται, γιατί κάποια μπορεί να είναι ευαίσθητα (κωδικοί) και δυστυχώς να μην έχουν κρυπτογραφηθεί. Με την ίδια λογική ο απλός χρήστης δεν έχει το δικαίωμα να στέλνει κατευθείαν πακέτα Ethernet, γιατί ανοίγει στην ουσία μια πίσω πόρτα για επιθέσεις DoS σε άλλα μηχανήματα του δικτύου.

Σημαντικό βιβλίο αναφοράς για τον δικτυακό προγραμματισμό είναι το [29]. Κάθε λειτουργικό έχει το δικό του interface για την αποστολή Ethernet πακέτων. Το BSD έχει το BSD Packet Filter (BPF) , το SVR4 το Datalink Provider Interface (DLPI), το Linux έχει τα SOCK_PACKET και PF_PACKET. Γεφυροποιός σε όλο αυτό το χάος είναι η βιβλιοθήκη libpcap που χρησιμοποιείται από το πρόγραμμα tcpdump. Έχει ένα σύνολο από συναρτήσεις που γεφυρώνουν στην ουσία τα διαφορετικά interfaces, καθιστώντας όμως τον προγραμματισμό πιο περίπλοκο. Στη δική μας περίπτωση, αφού η

ανάπτυξη γίνεται σε Linux επιλέξαμε να χρησιμοποιήσουμε το διαθέσιμο interface του PF_PACKET[1], γιατί ο προγραμματισμός του μοιάζει πάρα πολύ με τον προγραμματισμό του TCP/IP.

Τα βήματα που πρέπει να ακολουθήσουμε είναι:

1. Δημιουργία διεπαφής socket με την κλήση της συνάρτησης socket.
2. Κλήση της συνάρτησης bind για την λήψη πακέτων μόνο ενός συγκεκριμένου πρωτοκόλλου.

Ο κώδικας που υλοποιεί τα παραπάνω είναι:

```
sock = socket( PF_PACKET , SOCK_RAW , htons(aoe_ether_type));

/* bind the socket to an interface, I need to specify
* only the sll_family sll_protocol and sll_ifindex */
sock_addr.sll_family=AF_PACKET;
sock_addr.sll_protocol=htons(aoe_ether_type);
sock_addr.sll_ifindex=interface.ifr_ifindex;
bind( sock ,(const struct sockaddr *)&sock_addr
        , sizeof(struct sockaddr_ll);
```

Με την εντολή socket ανοίγουμε ένα καινούργιο socket τύπου PF_PACKET (μπορούμε να γράφουμε και να διαβάζουμε κατευθείαν από το datalink επίπεδο), που έχει χαρακτηριστικό SOCK_RAW (ό,τι γράφουμε μεταδίδεται κατευθείαν) και μπορούμε να λαμβάνουμε μόνο τα πακέτα που έχουν κωδικό πρωτοκόλλου αυτό της τρίτης παραμέτρου. Με την εντολή bind έχουμε την δυνατότητα να "δέσουμε" το socket και να λαμβάνουμε τα πακέτα μόνο από κάποιο συγκεκριμένο interface. Η χρήση του παραπάνω είναι εμφανής καθώς δεν χρειάζεται να επεξεργαζόμαστε όλη την κίνηση που περνά από όλες τις δικτυακές κάρτες του συστήματος, αλλά θέλουμε να λαμβάνουμε μόνο τα πακέτα που έχουν συγκεκριμένο κωδικό και προέρχονται από κάποια συγκεκριμένη κάρτα.

5.2 Προγραμματιστικό Μοντέλο GM

Το GM είναι ένα σύστημα επικοινωνίας για το Myrinet που βασίζεται στην μετάδοση μηνυμάτων. Σκοπός του είναι να προσφέρει επικοινωνία, η οποία να επιβαρύνει στο ελάχιστο τον επεξεργαστή του συστήματος, ο κώδικας να είναι μεταφέρσιμος, η μετάδοση να παρουσιάζει χαμηλή καθυστέρηση και να έχουμε τη μέγιστη δυνατή διαμεταγωγή δεδομένων. Για να επιτευχθούν τα παραπάνω,

[1] εκτελέστε "man 7 packet" για να δείτε περισσότερες πληροφορίες σχετικά με τον προγραμματισμό του

το ίδιο το σύστημα ανα λαμβάνει τον μεγαλύτερο επεξεργαστικό φόρτο της επικοινωνίας κάνοντας χρήση του επεξεργαστή που διαθέτει η ίδια η κάρτα δικτύου. Για να εκμεταλλευτεί μάλιστα τις μηχανές DMA του επεξεργαστή, απαιτεί το προς μεταφορά μηνύμα να βρίσκεται στη μνήμη και όχι να έχει μεταφερθεί σε κάποιο αρχείο swap.

Όλα τα παραπάνω χαρακτηριστικά καθιστούν τον προγραμματισμό σε GM τελείως διαφορετικό απ' ό,τι έχουμε συνηθίσει στον προγραμματισμό με χρήση Sockets. Πλέον όλη η επικοινωνία θα πραγματοποιείται μόνο με την χρήση των συναρτήσεων του GM API, που είναι ένας αρκετά αυστηρός μηχανισμός αποστολής μηνυμάτων (message passing system). Κατηγοριοποιεί το μηνύματα κατά τις παραμέτρους *size* (μέγεθος) και priority (προτεραιότητα). Έτσι μόνο εάν ο αποστολέας και ο παραλήπτης έχουν κάνει αντίστοιχες αιτήσεις, με τις παραπάνω παραμέτρους ίδιες, θα είναι επιτυχής η αποστολή του μηνύματος. Μήνυμα μικρότερου *size* δεν θα παραδοθεί αν υπάρχει διαθέσιμος καταχωρητής μεγαλύτερης τιμής *size*, ούτε θα υπάρξει επιτυχής μετάδοση αν υπάρχει διαθέσιμος καταχωρητής ίδιας τιμής *size*, αλλά διαφορετικής τιμής priority.

Παρακάτω παρατίθενται κάποιοι όροι που θα χρησιμοποιηθούν στην περιγραφή που ακολουθεί.

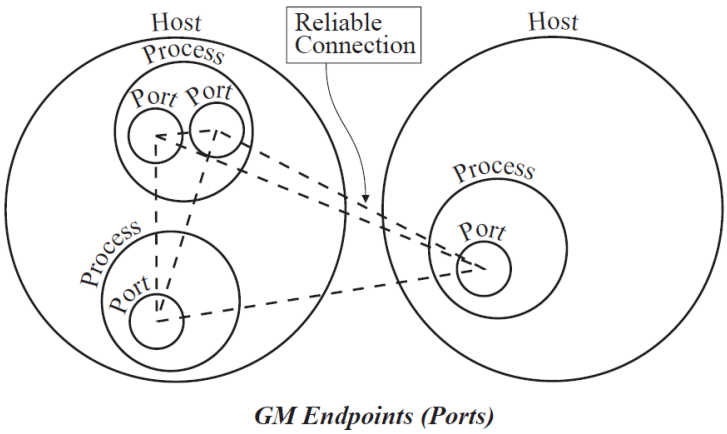
- ***length:*** Είναι το μέγεθος του μηνύματος σε bytes. Όταν αναφερόμαστε στο μήκος του καταχωρητή λήψης του μηνύματος, τότε είναι ο αριθμός των bytes που με ασφάλεια μπορούν να αποθηκευτούν σε αυτόν.

- ***size:*** Η τιμής *size* ενός μηνύματος είναι ένα ακέραιος μεγαλύτερος ή ίσος της τιμής $\log_2(length+8)$ με *length* να είναι το *μήκος* του μηνύματος.

  Το *size* ενός καταχωρητή λήψης είναι ένας θετικό αριθμός μικρότερος ή ίσος τις τιμής $\log_2(length+8)$. Έτσι ένας καταχωρητής μεγέθους *size* πρέπει να έχει *μήκος* τουλάχιστον $2^{size}-8$. Η συνάρτηση gm_min_size_for_length(*length*) μπορεί να χρησιμοποιηθεί για τον υπολογισμό τις ελάχιστης αναγκαίας τιμής του *size*, και η συνάρτηση gm_max_length_for_size(*size*) μπορεί αντίστοιχα να χρησιμοποιηθεί για τον υπολογισμό της μέγιστης τιμής του *length*.

5.2.1 Διεπαφές GM - ports

Το GM παρέχει συγκεκριμένες διεπαφές (ports) που εξασφαλίζουν την μετάδοση και την σειρά παράδοσης των αποτελουμένων μηνυμάτων, παρέχοντας δύο επίπεδα προτεραιότητας. Ο τύπος επικοινωνίας είναι ασυνδεσιστρεφής (connectionless) υπό την έννοια ότι, δε χρειάζεται να δημιουργηθεί κάποια σύνδεση για να αποσταλεί το μήνυμα, αλλά αρκεί ο πελάτης να ετοιμάσει το



Σχήμα 5.1: Διεπαφές επικοινωνίας - Ports

μήνυμά του, και να το στείλει σε οποιαδήποτε διεπαφή στο δίκτυο. Η σειρά των μηνυμάτων εξασφαλίζεται μόνο για τα μηνύματα που έχουν την ίδια προτεραιότητα και στέλνονται από την ίδια διεπαφή, και έχουν κοινή διεπαφή παραλήπτη. Μηνύματα με διαφορετικές προτεραιότητες δεν εμποδίζουν το ένα την μετάδοση του άλλου. Έτσι είναι δυνατό κάποιο μήνυμα χαμηλής προτεραιότητας να "προσπεράσει" ένα υψηλής προτεραιότητας.

5.2.2 Αρχικοποίηση-Τερματισμός

Για να μπορέσουμε να "ανοίξουμε" μια καινούργια διεπαφή (μετά την αρχικοποίηση με την κλήση της gm_init()) καλούμε την παρακάτω συνάρτηση:

```
gm_status_t gm_open( struct gm_port ** p, unsigned int unit,
unsigned int port, const char * port_name,enum gm_api_version
version)
```

Πεδίο	Σημασία
struct gm_port **port	Δείκτης προς την κατάσταση του port
int unit	Ο αριθμός της κάρτας στο σύστημα πχ myri0
int port	Ο αριθμός του port, υπάρχουν 8 και διαθέσιμοι προς χρήση είναι οι 2 και 4-7. Οι υπόλοιποι είναι δεσμευμένοι από το σύστημα
int port_name	String για λόγους αποσφαλμάτωσης
int gm_api_version	Έκδοση του χρησιμοποιούμενου GM API

Πριν από την έξοδό μας από το πρόγραμμα συστήνεται να κλείσουμε την διεπαφή και να απελευθερώσουμε όσους πόρους της κάρτας έχουμε δεσμεύσει

με την κλήση των παρακάτω συναρτήσεων.

```
void gm_close(struct gm_port *p)
void gm_finalize(void)
```

5.2.3 Δέσμευση Μνήμης

Το μεγαλύτερο μήνυμα που μπορεί να να στείλει ή να λάβει το GM περιορίζεται στα $2^{31} - 1$ bytes. Παρ' όλα ταύτα, επειδή οι καταχωρητές που θα χρησιμοποιηθούν για την λήψη ή την παραλαβή του μηνύματος πρέπει να βρίσκονται σε μνήμη, που να είναι προσβάσιμη από τον μηχανισμό DMA, το μέγιστο μήνυμα που μπορεί να σταλεί, περιορίζεται από την φυσική μνήμη του συστήματος. Οι εφαρμογές GM θα μπορέσουν να χειριστούν κατευθείαν μνήμη με την χρήση των συναρτήσεων [2]:

```
void* gm_dma_malloc(struct gm_port * p, gm_size_t length)
void gm_dma_free(gm_port_t *p, void * addr)
```

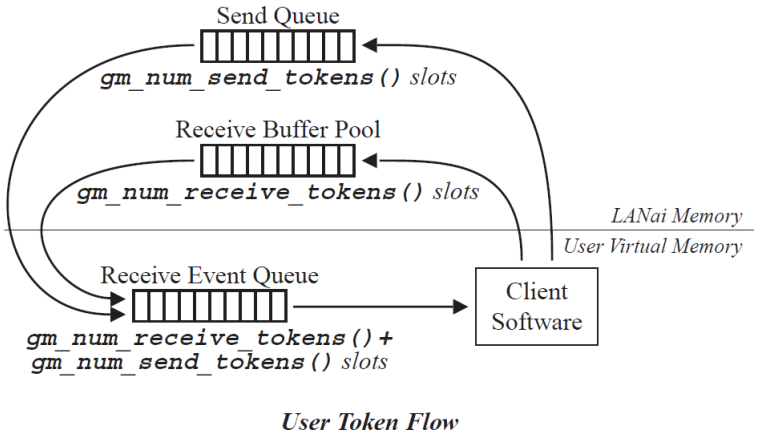
Για ποιο "έξυπνα" προγράμματα με μεγάλες απαιτήσεις σε μνήμη δίνεται η δυνατότητα να κάνουν "pin" και "unpin" την μνήμη με την κλήση των συναρτήσεων αρκεί να το υποστηρίζει το λειτουργικό σύστημα:

```
gm_status_t gm_register_memory_ex(gm_port_t * p,
                                  void *ptr,
                                  gm_size_t length,
                                  void * pvma)
gm_status_t gm_deregister_memory(struct gm_port * p,
                                 void* ptr,
                                 gm_size_t length)
```

5.2.4 Μηχανισμός Tokens

Όπως έχουμε περιγράψει παραπάνω αν και οι κάρτες Myrinet έχουν μια πολύ ισχυρή μονάδα επεξεργασίας, οι διαθέσιμοι λοιποί πόροι που διαθέτει είναι εκ των πραγμάτων περιορισμένοι. Το GM για να μπορέσει να ελέγξει του διαθέσιμους πόρους που χρησιμοποιούνται για την υλοποίηση των ουρών μετάδοσης και λήψης των μηνυμάτων, χρησιμοποιεί τον μηχανισμό των tokens (κερμάτων). Μόνο αν έχει στην διάθεσή της η εφαρμογή διαθέσιμο token τότε μόνο είναι σε θέση να λάβει ή να στείλει κάποιο μήνυμα. Η εφαρμογή καταναλώνει ένα token, όταν καλεί τις συναρτήσεις που θα δούμε παρακάτω για την αποστολή και την λήψη των μηνυμάτων, και της επιστρέφονται πίσω όταν η διαδικασία έχει ολοκληρωθεί. Η εφαρμογή είναι υπεύθυνη για τον

[2] Προσοχή πρέπει να δοθεί στο γεγονός ότι η δεσμευμένη μνήμη σχετίζεται με συγκεκριμένο port. Αυτή η μνήμη δε μπορεί να χρησιμοποιηθεί από κάποιο άλλο port αν δε γίνει προηγουμένως "registered" όπως θα δούμε παρακάτω



Σχήμα 5.2: Λειτουργία μηχανισμού Tokens

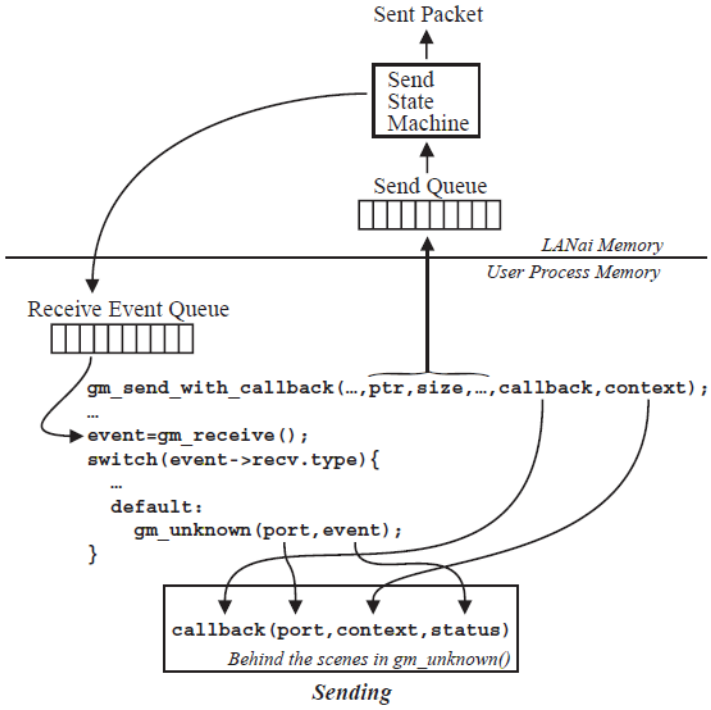
ελέγχο των διαθέσιμών της tokens. Μπορούμε να δούμε πόσα tokes έχουμε διαθέσιμα για αποστολή και λήψη, με την κλήση των παρακάτω συναρτήσεων:

```
unsigned int gm_num_send_tokens(struct gm_port * p)
unsigned int gm_num_receive_tokens(struct gm_port * p)
```

5.2.5 Αποστολή μηνύματος

Ο τρόπος του μηχανισμού αποστολής μηνυμάτων φαίνεται στο σχήμα 5.3 και αναφέρεται ουσιαστικά στην λειτουργία της συνάρτησης:

```
void gm_send_with_callback(struct gm_port *    p,
                           void *    message,
                           unsigned int    size,
                           gm_size_t    len,
                           unsigned int    priority,
                           unsigned int    target_node_id,
                           unsigned int    target_port_id,
                           gm_send_completion_callback_t    callback,
                           void *    context)
```



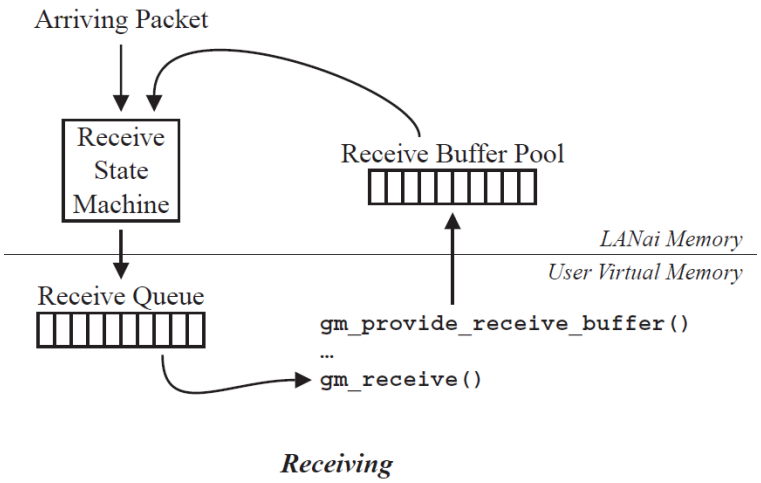
Σχήμα 5.3: Μηχανισμός αποστολής μηνύματος

Πεδίο	Σημασία
p	Η port μέσω της οποίας θα σταλεί το μήνυμα
message	Το προς αποστολή μήνυμα
size	Η τιμή *size* του μηνύματος
len	Το μήκος σε bytes του μηνύματος
priority	Η προτεραιότητα
target_node_id	Ο αριθμός του κόμβου του παραλήπτη
target_port_id	Ο αριθμός του port στον παραλήπτη
callback	Η συνάρτηση που θα κληθεί με το τέλος της αποστολής
context	Η παράμετρος που θα περαστεί στη συνάρτηση callback

Το σχήμα 5.3 είναι σαφές του τρόπου που γίνεται η αποστολή του μηνύματος. Δεδομένου ότι έχουμε στην διάθεσή μας ένα token καλούμε την συνάρτηση αποστολής. Αυτή εισάγει την αίτησή μας στην αντίστοιχη ουρά. Με την ολοκλήρωση της αποστολής έχουμε ένα καινούργιο γεγονός στην ουρά συμβάντων του τύπου GM_SEND_EVENT. Περνούμε το συμβάν αυτό στην συνάρτηση gm_unkown() που με την σειρά της αυτή καλεί την συνάρτηση callback που της είχαμε διαθέσει. Με την κλήση της συνάρτησης callback μας

επιστρέφεται στην ουσία το token που είχαμε διαθέσει για την αποστολή του μηνύματος. Μέσα στην συνάρτηση callback είμαστε σε θέση να ελέγχουμε την κατάσταση της απεσταλμένης αίτησής μας. Κανονικά οποιαδήποτε κατάσταση διαφορετική από το GM_SUCCESS θα μπορούσε να θεωρηθεί ως σφάλμα. Υπάρχει όμως μια πληθώρα καταστάσεων που είναι σε θέση να μας περιγράψουν όλα τα πιθανά σφάλματα που έχουν συμβεί, επιτρέποντάς μας να επαναλάβουμε την μετάδοση του μηνύματος, αν αυτό είναι δυνατόν ή να βρούμε κάποιο προγραμματιστικό σφάλμα.

5.2.6 Λήψη μηνύματος



Σχήμα 5.4: Μηχανισμός λήψης μηνύματος

Όπως είδαμε και ο μηχανισμός της λήψης των μηνυμάτων ελέγχεται και αυτός μέσω των tokens. Για να μπορέσουμε να λάβουμε ένα καινούργιο μήνυμα θα πρέπει να παρέχουμε στο σύστημα τον αντίστοιχο καταχωρητή για να αποθηκεύσει το μήνυμα. Αυτό γίνεται με την κλήση της παρακάτω συνάρτησης:

```
void gm_provide_receive_buffer_with_tag(struct gm_port * p,
                                        void *   ptr,
                                        unsigned int   size,
                                        unsigned int   priority,
                                        unsigned int   tag)
```

Πεδίο	Σημασία
p	Το port στο οποίο παρέχεται ο buffer
ptr	Η διεύθυνση του buffer
size	Η παράμετρος *size* του μηνύματος
priority	Η προτεραιότητα του μηνύματος
tag	Τιμή δείκτης του τύπου του μηνύματος που λάβαμε

Καλώντας την παραπάνω συνάρτηση "ξοδεύουμε" και ένα token που θα λάβουμε πίσω με την κλήση της gm_receive. Πρέπει επίσης να έχουμε εξασφαλίσει ότι το μέγεθος του καταχωρητή για μέγεθος *size* είναι τουλάχιστον gm_max_lenght_for_size(). Μέγεθος μεγαλύτερο του παραπάνω δεν έχει οποιαδήποτε χρήση στο σύστημα. Επίσης είμαστε σε θέση να παρέχουμε περισσότερους του ενός καταχωρητές του ιδίου μεγέθους και προτεραιότητας αν το θεωρούμε σκόπιμο. Όταν το σύστημα λάβει κάποιο μήνυμα με τα ίδια χαρακτηριστικά μεγέθους και προτεραιότητας τότε θα το αποθηκεύσει σε κάποιον από τους καταχωρητές που του έχουμε παράσχει

Αφού έχουμε διαθέσει στο σύστημα τους κατάλληλους καταχωρητές για να μπορέσουμε να ενημερωθούμε για την λήψη κάποιου μηνύματος ή για την ολοκλήρωση κάποιας αποστολής, θα πρέπει να καλέσουμε μια συνάρτηση της οικογένειας gm*_receive*(). Μέσω αυτών των συναρτήσεων μπορούμε να δούμε την κατάσταση στην ουρά των γεγονότων (receive event queue). Αν έχουμε την λήψη κάποιου καινούργιου μηνύματος τότε θα έχουμε ένα καινούργιο γεγονός του τύπου GM_RECV_EVENT ή GM_HIGH_RECV_EVENT, που αναφέρονται στην άφιξη μηνύματος χαμηλής ή υψηλής προτεραιότητας αντίστοιχα. Αν λάβουμε κάποιο άλλο γεγονός τότε αυτό μπορούμε με ασφάλεια να το περάσουμε στην συνάρτηση gm_unkown(), γιατί είναι επί το πλείστον γεγονότα που αφορούν την εσωτερική λειτουργία του GM.

5.3 Υλοποίηση του AoE Server

```
struct Aoehdr {
        uchar dst[6];
        uchar src[6];
        u16 type;
        uchar flags;
        uchar error;
        u16 maj;
        uchar min;
        uchar cmd;
        u32 tag;
};
```

```
struct Ata {
        Aoehdr h;
        uchar aflag;
        uchar err;
        uchar sectors;
        uchar cmd;
        uchar lba[6];
        uchar resvd[2];
        uchar data[BUFF_SIZE];
};

struct Conf {
        Aoehdr h;
        u16 bufcnt;
        u16 firmware;
        uchar filler;
        uchar vercmd;
        u16 len;
        uchar data[BUFF_SIZE];
};
```

Οι δομές που παρατίθενται παραπάνω αναφέρονται στις επικεφαλίδες του πρωτοκόλλου Ata over Ethernet όπως αυτό περιγράφεται στην παράγραφο 2.2. Στη παράγραφο 3.1.2 είδαμε την αρχιτεκτονική ένας AoE server.

Για να φτιάξουμε ένα υποσύστημα για το vRAID όπως αυτά του 4.1, στον υπάρχοντα server θα πρέπει να προσθέσουμε ένα μηχανισμό που θα μας επιτρέπει να εντοπίσουμε αν "συνδέεται" κάποιος καινούργιος client και να του αντιστοιχήσουμε έναν αριθμό connection όπως απαιτεί το σύστημα vRAID.

5.4 Υλοποίηση του myL3

Το πρωτόκολλο myL3 έχει περιγραφεί αναλυτικά στις παραγράφους 2.3 και 3.2. Η υλοποίησή τους προσπαθήσαμε να είναι τόσο απλή όσο και το πρωτόκολλο το ίδιο. Σκοπός μας δεν είναι να παράγουμε ένα πιο πολύπλοκο πρωτόκολλο, γιατί στην ουσία θα συγκλίναμε σε ήδη έτοιμες λύσεις (TCP/IP).

5.4.1 Υλοποίηση του myL3 Server

Ο server απομένει αρκετά "χαζός". Διεκπεραιώνει μόνο εντολές χωρίς να ενδιαφέρεται για την επιτυχία της αποστολής του μηνύματός του. Η πληθώρα των νημάτων έγινε με την ελπίδα ότι μια τέτοια υλοποίηση θα μπορεί να διεκπεραιώνει ποιο γρήγορα τις διάφορες αιτήσεις.

5.4.2 Υλοποίηση του myL3 Client

Ο client από την μεριά του είναι αυτός που αναλαμβάνει όλο το φορτίο της παραγωγής των αιτήσεων και του ελέγχου της σωστής μετάδοσης των μηνυμάτων. Η διαδρομή που θα ακολουθήσει μια αίτηση από τα υψηλότερα επίπεδα του vRAID είναι:

- **Παραγωγή μηνυμάτων προς αποστολή.** Με τη λήψη μιας καινούργιας αίτησης από την ουρά εισόδου, θα πρέπει να παραχθούν τα αντίστοιχα μηνύματα του myL3. Για μηνύματα που αναφέρονται σε εντολές εγγραφής ή ανάγνωσης, μηνύματα όπου θα πρέπει να μεταφερθούν δεδομένα από ή προς τον server, υπάρχει ο περιορισμός του MTU του Ethernet. Αν λοιπόν η αίτηση θα πρέπει να μεταφέρει δεδομένα περισσότερα απ' ότι το διαθέσιμο MTU, τότε θα διασπαστεί σε περισσότερα του ενός διαδοχικά μηνύματα myL3, τα οποία θα εκτελεστούν μεμονωμένα στον server. Κάθε κομμάτι θα έχει προσαρμοσμένες τις τιμές length και offset για να είναι δυνατή η άμμεση εκτέλεσή του. Για μηνύματα που δεν μεταφέρουν δεδομένα, και εκ των πραγμάτων "χωράνε" σ' ένα πακέτο Ethernet, για κάθε αίτηση του vRAID αντιστοιχεί και ένα πακέτο myL3. Τα έτοιμα πακέτα εισέρχονται σε μια λίστα όπου και περιμένουν την αποστολή τους.

  Η αίτηση vRAID εισάγεται στην ουρά αναμονής, ενώ παράλληλα δημιουργείται μια αντίστοιχη καταχώρηση κατακερματισμένου μηνύματος. Σε αυτή θα αποθηκευτούν προσωρινά οι επιμέρους απαντήσεις μιας αίτησης vRAID, μέχρι να συμπληρωθεί ο αναμενόμενος αριθμός τους.

- **Αποστολή μηνυμάτων.** Για την αποστολή των μηνυμάτων θα πρέπει να υλοποιηθεί κάποιος έλεγχος ροής, γιατί οι διαθέσιμοι από το λειτουργικό buffers για την αποστολή και την λήψη των μηνυμάτων είναι περιορισμένοι. Αν το αποφύγουμε τότε εκ των πραγμάτων θα έχουμε μεγάλη πληθώρα από χαμένα πακέτα, με αποτέλεσμα να στέλνουμε ξανά και άσκοπα μεγάλο μέρος της πληροφορίας.

  Ο μηχανισμός βασίζεται στην ιδέα του παραθύρου όπως στο TCP/IP. Έτσι ορίζουμε έναν μέγιστο αριθμό από πακέτα που μπορεί να είναι ήδη απεσταλμένα, και βρίσκονται στην αντίστοιχη λίστα αποστολής. Όταν η λίστα αποστολής είναι άδεια κατά το ήμισυ τότε μπορούμε να στείλουμε καινούργια πακέτα μέχρι να φτάσουμε στο μέγιστο επιτρεπόμενο αριθμό. Στη παρούσα υλοποίηση, το μέγεθος του παραθύρου ορίζεται στατικά κατά την έναρξη της λειτουργίας. Το μέγεθός του έχει προκύψει πειραματικά, με παράλληλη ρύθμιση του συστήματος Linux όπου

αυξήσαμε τους αντίστοιχους buffers, έτσι ώστε να μην έχουμε μεγάλο αριθμό από χαμένο πακέτα.

Έγιναν προσπάθειες ώστε αυτή η διαδικασία να γίνεται αυτόματα, και το σύστημα να βρίσκει πιο είναι το βέλτιστο μέγεθος του παραθύρου, αλλά δεν καρποφόρησαν. Η μέθοδος που ακολούθησα ήταν: μετά την επιτυχή λήψη όλων των απαντήσεων των μηνυμάτων, που είχαν σταλεί για μέγεθος παραθύρου a πακέτων, να το αυξάνω κατά ένα. Το μέγεθος όμως του μηνύματος είναι αρκετά μεγάλο (δύο sectors), με αποτέλεσμα να εισάγει μεγάλα σφάλματα στο σύστημα, και να έχουμε πολλά χαμένα πακέτα.

Κατά την αποστολή του στο μήνυμα επισυνάπτεται μια χρονική σφραγίδα, που μας επιτρέπει να αποφασίσουμε την αποστολή του εκ νέου αν δεν λάβουμε απάντηση μέσα στο όρια του υπολογιζόμενου RTT.

- **Λήψη μηνυμάτων.** Με τη λήψη ενός μηνύματος επιβεβαίωσης γίνεται αναζήτηση στη λίστα αποστολής για την εύρεση αντίστοιχης αίτησης myL3. Στην περίπτωση που δε βρεθεί, τότε πρόκειται για κάποιο μήνυμα όπου έγινε εκ νέου η αποστολή της αίτησης λόγω λήξης του χρόνου αναμονής, με την υποψία ότι είχαμε απώλεια της αρχικής αίτησης, χωρίς όμως αυτό να συμβεί πραγματικά. Έχουμε λοιπόν μια διπλή απάντηση που απλώς δεν αγνοούμε.

  Στην συνέχεια υπολογίζεται η καινούργια μέση τιμή του RTT, με τη χρήση της χρονικής χρονικής σφραγίδας που έχει η αίτηση.

  Παράλληλα βρίσκεται η αντίστοιχη καταχώρηση κατακερματισμένου μηνύματος. Αδυναμία εύρεσής του αποτελεί σφάλμα που αναφέρεται. Εισάγεται η απάντηση, και αν έχει συμπληρωθεί ο αριθμός των αναμενόμενων απαντήσεων, το κατακερματισμένο μήνυμα είναι πλήρες. Αφαιρείται η αντίστοιχη αίτηση vRAID και αντιγράφονται τα δεδομένα από την κατακερματισμένο μήνυμα στην αίτηση vRAID, και η συνολική απάντηση προωθείται στα υψηλότερα επίπεδα του vRAID.

- **Επανάληψη αποστολής.** Επανάληψη αποστολής υπάρχει σε δύο περιπτώσεις. Αν δεν λάβουμε κάποια καινούργια απάντηση μέσα στο χρόνο ενός RTT, ψάχνουμε στην λίστα αποστολής και στέλνουμε ξανά τα μηνύματα, για τα οποία δεν έχουμε λάβει απάντηση μέσα σε χρόνο RTT. Μετά από την λήψη κάθε μηνύματος γίνεται πάντα έλεγχος στη λίστα αποστολής για να εντοπιστούν τα μηνύματα για τα οποία δεν έχουμε λάβει απάντηση μέσα σε χρόνο RTT.

5.4.2.1 Παρατηρήσεις

Όπως θα δούμε παρακάτω, και στις μετρήσεις που θα κάνουμε, η επίδοση του όλου συστήματος είναι αρκετά χαμηλή. Η αιτία είναι ότι το δίκτυο αποτελεί στην ουσία την στενωπό του συστήματος. Οι εντολές I/O που πραγματοποιούμε με κάθε μήνυμα είναι αρκετά μικρές, αφού το μέγεθος των δεδομένων που μπορούμε να μεταφέρουμε περιορίζεται από το MTU, και ο δίσκος παρουσιάζει χαμηλές επιδώσεις για "μικρά" I/O.

Έγιναν προσπάθειες ο server να ομαδοποιήσει της εντολές I/O με βάση την αρχική αίτηση vRAID, είδαμε όμως ότι κάτι τέτοιο δεν ήταν υλοποιήσιμο. Τα προβλήματα ήταν δύο. Οι απώλειες μηνυμάτων που μπορεί να συμβούν εμποδίζουν τον εύκολο εντοπισμό, με βάσει τα μηνύματα myL3, της αντίστοιχης συνολικής αίτησης vRAID. Μπορεί δηλαδή να έχουμε την απώλεια του πρώτου κατά σειρά μηνύματος myL3 μιας ομάδας μηνυμάτων που ανήκουν στην ίδια αίτηση vRAID. Αποτέλεσμα είναι να μην μπορούμε να καθορίσουμε ποια είναι η αίτηση vRAID, ούτε να εγγυηθούμε ότι έχουν φτάσει όλα τα αναμενόμενα μηνύματα όταν πραγματοποιήσουμε την εντολή I/O. Επίσης θα χρειαζόταν να εισάγουμε επιπλέον μηνύματα επιβεβαίωσης. Έτσι εκτός από τα μηνύματα που επιβεβαιώνουν την επιτυχή λήψης των επιμέρους πακέτων, θα χρειαζόταν να στείλουμε και επιβεβαίωση της επιτυχίας της εντολής I/O. Το γεγονός όμως ότι το μέσο μετάδοσης δεν είναι ασφαλές σημαίνει ότι η επιβεβαίωσή μας μπορεί να μην φτάσει. Τα γεγονός αυτό επιβαρύνει επιπλέον τον client που θα πρέπει να είναι σε θέση να ξαναστείλει όλα τα επιμέρους πακέτα αν δεν λάβει μήνυμα επιβεβαίωσης της εντολής I/O. Αν λάβουμε υπόψη ότι η αιτία να χαθεί το μήνυμα συνήθως είναι η υπερφόρτωση του δικτύου καταλαβαίνουμε ότι μια τέτοια διαδικασία θα εισάγει μόνο άσκοπη κίνηση και επαναλήψεις εντολών I/O, παρά θα βελτιώσει την συνολική επίδοση υπό κανονικές ρυθμίσεις του συστήματος.

Τέλος είδαμε ότι υπάρχει η πιθανότητα να χαθεί κάποια επιβεβαίωση μηνύματος, γεγονός που θα σημάνει την επανάληψης της αποστολής της αρχικής αίτησης myL3. Στην περίπτωση που πρόκειται για κάποια εντολή I/O τότε θα έχουμε απλώς την άσκοπη επανάληψή της χωρίς να εισάγεται κάποιο σφάλμα. Στην περίπτωση όμως που έχουμε κάποια εντολή κλειδώματος, η απώλεια της επιβεβαίωσης θα σημάνει την εσφαλμένη αποστολή της εκ νέου. Κάτι τέτοιο μπορεί να οδηγήσει σε deadlock με την υπάρχουσα υλοποίηση του μηχανισμού κλειδωμάτων στον server, γιατί θα προσπαθήσουμε στην ουσία να κλειδώσουμε κάτι που κλειδώσαμε πριν από λίγο εμείς οι ίδιοι. Μια λύση θα μπορούσε να είναι η μεταφορά του μηχανισμού κλειδωμάτων στον vRAID controller, κάτι τέτοιο όμως εισάγει το πρόβλημα του συγχρονισμού του μηχανισμού των κλειδωμάτων όταν υπάρχουν διατάξεις με περισσότερους του ενός controllers.

5.5 Υλοποίηση σε GM API

Στη παράγραφο 5.2 περιγράψαμε γενικά το προγραμματιστικό μοντέλο του GM, ας δούμε τώρα πώς μπορούμε να το χρησιμοποιήσουμε για την ανάπτυξη εφαρμογών βασισμένες σε αυτό.Παρακάτω θα περίγράψουμε τον τρόπο λειτουργίας που δώσαμε σχηματικά στη παράγραφο 3.3. Αρχικά θα αναφέρουμε τα κοινά στοιχεία του server και του client και στη συνέχεια θα αναφερθούμε στις επιμέρους λεπτομέρειες.

Όπως είδαμε το GM προσφέρει περιορισμένο αριθμό από διεπαφές επικοινωνίας (gm_ports), έτσι ώστε να μας υποχρεώνει, αν θέλουμε να έχουμε περισσότερες από μια "συνδέσεις" να πρέπει να υλοποιήσουμε μια αρχιτεκτονική δύο επιπέδων, όπου το ένα θα έχει ως αποκλειστική ευθύνη την αποστολή και την λήψη μηνυμάτων, με την υποχρέωση να επιστρέφει το μύνημα πίσω στην σωστή σύνδεση. Εδώ θα πρέπει να επισημάνουμε ότι στο GM δεν υπάρχει η έννοια της σύνδεσης, αφού η λειτουργία του περιορίζεται στην αποστολή και τη λήψη μηνυμάτων, χωρίς να δημιουργεί συνδέσεις με την έννοια που τις συναντάμε στα Sockets.

Στις υλοποιήσεις μας θα χρησιμοποιήσουμε 2 gm_port το ένα για την λήψη μηνυμάτων και το άλλο για την αποστολή. Παράγοντας που συνετέλεσε σε αυτή την απόφαση είναι το γεγονός ότι το GM δεν επιτρέπει την ασφαλή ανάπτυξη εφαρμογών με την χρήση νημάτων Οι συναρτήσεις του δεν μας εξασφαλίζουν ότι θα πραγματοποιήσουν κάθε κλήση τους με ατομικό τρόπο εσωτερικά. Έτσι αναλαμβάνουμε εμείς την υποχρέωση ότι, όποιες κλήσεις μπορεί να μεταβάλλουν την κατάσταση κάποιου gm_port, να γίνονται μεμονωμένα. Επίσης είδαμε ότι κάποιες συναρτήσεις του API μπορεί να αδρανοποιήσουν το εκτελούμενο νήμα στην αναμονή τους για κάποιο γεγονός από το gm_port, γεγονός που μπορεί να προκαλέσει deadlock στην εφαρμογή μας, αν έχουμε κλειδώσει από πριν κάποιον mutex, και δεν φροντίσουμε να τον ελευθερώσουμε πριν μπούμε σε κατάσταση αναμονής. Αυτοί οι δύο λόγοι και μόνο, αρκούν, για να μας οδηγήσουν στην χρήση δύο gm_ports.

Η αποστολή και η λήψη μηνυμάτων μπορεί να γίνει μόνο σε καταχωρητές που βρίσκονται στην φυσική μνήμη και η διεύθυνσή τους μπορεί να υπολογιστεί από την μηχανή DMA της κάρτας Myrinet. Για να πληρεί αυτές τις προδιαγραφές η μνήμη, μετά την δέσμευσή της, θα πρέπει να γίνει pin down, μια διαδικασία ποιο δαπανηρή από την αποστολή ή την λήψη κάποιου μηνύματος. Επίσης είδαμε ότι η ο έλεγχος ροής γίνεται με την χρήση tokens, όπου δεν μπορούμε να λάβαμε ούτε να αποστείλουμε κάποιο μήνυμα αν δεν διαθέτουμε κάποιο token. Για τους παραπάνω λόγους κατά την έναρξη των δύο προγραμμάτων λαμβάνουμε τον αριθμό των διαθέσιμων tokens λήψης[3], τα

[3]Είδαμε στην πράξη ότι τα διαθέσιμα tokens για την λήψη μηνυμάτων είναι περισσότερα

κατανέμουμε στους διάφορους τύπους μηνυμάτων που περιμένουμε, και στην συνέχεια δεσμεύουμε μνήμη για ισάριθμους buffers, τόσο για λήψη όσο και για αποστολή. Κάθε buffer συσχετίζεται όπως είδαμε στη παράγραφο 5.2.3 με συγκεκριμένο gm_port. Τέλος τοποθετούμε τους buffers σε λίστες ή ουρές έτσι ώστε να είμαστε σε θέση να τους διαχειριστούμε.

Στο πρόγραμμά μας έχουμε τρεις τύπους μηνυμάτων. Τα μηνύματα ρύθμισης (configuration messages) που είναι υψηλής προτεραιότητας και τα οποία τα χρησιμοποιούμε για την αρχικοποίηση, τόσο του server, όσο και του client όπως φαίνονται παρακάτω:

```
struct  conf_rsp {
        uint64_t        mediasize;
        uint8_t         error;
};
struct  conf_req{
        uint32_t        sector_size;
        uint32_t        chunk_size;
        uint8_t         slack;/* in order to have same size with rsp */
};
```

Τα μηνύματα επικεφαλίδες, κανονικής προτεραιότητας, που περιέχουν πληροφορίες σχετικά με την εντολή που θέλουμε να εκτελεστεί (εγγραφή,ανάγνωση,κτλ) στο δίσκο.

```
struct header {
        uint32_t        connection;
        uint32_t        sequence;
        uint8_t         command;
        uint64_t        offset;
        uint32_t        length;
        uint8_t         error;
};
```

Τέλος υάρχουν τα μηνύματα δεδομένων, κανονικής προτεραιότητας, που περιέχουν τα δεδομένα προς εγγραφή ή που έχουν προέλθει από κάποια ανάγνωση. Για τα τελευταία δεν χρησιμοποιούμε κάποια επικεφαλίδα στην αρχή τους. Το GM εξασφαλίζει ότι μηνύματα της ίδιας προτεραιότητας (ανεξαρτήτως μεγέθους) που έχουν τον ίδιο αποστολέα θα παραδοθούν με την σειρά αποστολής τους. Έτσι κάθε μήνυμα δεδομένων συσχετίζεται με την προηγούμενη επικεφαλίδα, που έφτασε από τον ίδιο αποστολέα. Απουσία επικεφαλίδας συνιστά σφάλμα για το σύστημα.

από τα αντίστοιχα για την αποστολή μηνυμάτων.

5.5.1 GM server

Η βασική δομή που χρησιμοποιείται κατά την αρχικοποίηση και την λειτουργία του server φαίνεται παρακάτω:

```
struct  myri_sdata{
        struct gm_port  *send_port;
        struct gm_port  *rcv_port;
        gm_u32_t        unit;
        int             sector_size;
        int             chunk_size;
        gm_u32_t        receive_tokens;
        gm_u32_t        send_tokens;
        gm_u32_t        rcv_conf_tokens;
        gm_u32_t        rcv_header_tokens;
        gm_u32_t        rcv_buffer_tokens;
        struct message_queue    *send_confs;
        struct message_queue    *send_headers;
        struct message_queue    *send_buffers;
        struct list             *rcv_confs;
        struct list             *rcv_headers;
        struct list             *rcv_buffers;
        struct message_queue    *out_queue;
        char    filename[MAX_FILE_PATH];
        int     filedes;
        gm_u64_t      mediasize;
        pthread_mutex_t mutex;
};
```

Πεδίο	Σημασία
send_port	GM port για την αποστολή μηνυμάτων
rcv_port	GM port για την λήψη μηνυμάτων
unit	Ο αριθμός της κάρτας δικτύου Myrinet όπως παρουσιάζεται στο σύστημα
sector_size	Το μέγεθος του sector που θα χρησιμοποιήσουμε στο σύστημα σε bytes
chunk_size	Το μέγιστο μέγεθος μηνύματος σε sectors
receive_tokens	Τα διαθέσιμα tokens για την λήψη μηνυμάτων
send_tokens	Τα διαθέσιμα tokens για την αποστολή μηνυμάτων
rcv_conf_tokens	Τα tokens που χρησιμοποιούνται για την λήψη μηνυμάτων ρύθμισης
rcv_header_tokens	Τα tokens που χρησιμοποιούνται για την λήψη επικεφαλίδων
rcv_buffer_tokens	Τα tokens που χρησιμοποιούνται για την λήψη δεδομένων
send_confs	Ουρά με τους διαθέσιμους buffers για την αποστολή μηνυμάτων ρύθμισης
send_headers	Ομοίως για τις επικεφαλίδες
send_buffers	Ομοίως για τα δεδομένα
rcv_confs	Λίστα με τους διαθέσιμους buffers για την λήψη μηνυμάτων ρύθμισης
rcv_headers	Ομοίως για τις επικεφαλίδες
rcv_buffers	Ομοίως για τα δεδομένα
out_queue	Ουρά μηνυμάτων προς αποστολή
filename	Το όνομα του αρχείου που χρησιμοποιούμε ως εικονικό δίσκο
filedes	Ο file descriptor του παραπάνω αρχείου
mediasize	Το μέγεθος τους αρχείου σε sectors
mutex	Mutex που χρησιμοποιείται για την ρύθμιση της αλληλουχίας της έναρξης των νημάτων του Server.

Πίνακας 5.1: Περιγραφή των πεδίων της δομής myri_sdata.

Στο σχήμα 3.7 είδαμε σχηματικά την δομή του server. Στον πίνακα 5.1 περιγράφονται τα πεδία του κεντρικού του struct, όπως αυτά μπορούν να προκύψουν και από την γενική περίγραφή που παραθέσαμε παραπάνω. Τα βήματα της λειτουργίας του server είναι

- **Αρχικοποίηση** Κατά την έναρξή του ο server περιμένει την αρχικοποίησή του. Οι τιμές που θα πρέπει να καθοριστούν από το σύστημα vraid είναι τα μεγέθη των sector_size και chunk_size. Ο πρώτος client που θα συνδεθεί με τον server θα είναι και αυτός που θα θέσει τις παραπάνω τιμές με την χρήση ενός μηνύματος ρύθμισης. Επόμενα μηνύματα ρύθμισης θα απαντηθούν μόνο χωρίς να επηρεάσουν τις παραμέτρους του.

  Μετά την αρχικοποίηση θα δεσμεύσει την απαραίτητη μνήμη για τη λήψη των επικεφαλίδων και των δεδομένων. Οι buffers που θα χρησιμοποιηθούν για την λήψη και την αποστολή θα είναι ισάριθμοι, αν και τα tokens αποστολής είναι λιγότερα. Η αιτία της παρακάτω απόφασης θα γίνει κατανοητή κατά την περίγραφή της λειτουργίας.

- **Λειτουργία Λήψης** Πλέον το σύστημα είναι έτοιμο για την λήψη μηνυμάτων. Να σημειώσουμε ότι παράλληλα με την δέσμευση των buffers λήψης καλέσαμε και την συνάρτηση gm_provide_receive_buffer_with_tag() που παρέχει τους buffers στο σύστημα.

  Για την λήψη κάποιου καινούργιου μηνύματος πρέπει να διαπιστώσουμε αν υπάρχει κάποιο αντίστοιχο γεγονός στην ουρά. Για να το πραγματοποιήσουμε αυτό καλούμε την συνάρτηση gm_blocking_receive_no_spin() που αν δεν έχουμε την λήψη καινούργιου μηνύματος επιστρέφει αμέσως με τύπο μηνύματος _GM_SLEEP_EVENT, που τον περνάμε στην συνάρτηση gm_unkown() και έτσι αδρανοποιούμε το νήμα λήψης μέχρι την άφιξη κάποιου μηνύματος.

  Αν όντως φτάσει κάποιο καινούργιο μήνυμα, τότε μπορούμε ξεχωρίσουμε τον τύπο του από την προτεραιότητά του, και από την τιμή TAG που είχαμε δώσει όταν παρείχαμε τον buffer στο GM. Αν είναι μήνυμα ρύθμισης, τότε ετοιμάζουμε κατάλληλη απάντηση και την αποστέλλουμε. Αν είναι επικεφαλίδα, τότε ανάλογα με την εντολή που θα εκτελέσει έχουμε δύο περίπτώσεις:

  1. Αν η εντολή για να εκτελεστεί θα πρέπει να συνοδεύεται από δεδομένα που ακολουθούν, τότε τοποθετούμε την επικεφαλίδα σε μια λίστα απ' όπου θα την πάρουμε στην συνέχεια με την άφιξη των δεδομένων.
  2. Αν πρόκειται για οποιαδήποτε άλλη εντολή, τότε προχωράμε αμέσως στην εκτέλεσή της.

  Τέλος άν έχουμε την άφιξη των δεδομένων, βρίσκουμε την αμέσως προηγούμενη επικεφαλίδα στην λίστα αναμονής και εκτελούμε την εντολή.

Κατά την εκτέλεση της εντολής λαμβάνουμε αρχικά τους buffers που θα χρειαστούμε για την αποστολή του μηνύματος από τις αντίστοιχες ουρές. Οι ουρές που χρησιμοποιούμε θα μπλοκάρουν μέχρι να υπάρχουν διαθέσιμοι buffers. Κάτι τέτοιο όμως είναι πολύ απίθανο να συμβεί, γιατί έχουμε φροντίσει οι buffers αποστολή να είναι ισάριθμοι με τους buffers λήψης. Μετά το τέλος της εκτέλεσης της εντολής τοποθετούμε τα παραγόμενα μηνύματα στην ουρά "εξόδου".

Μετά το τέλος της εκτέλεσης της εντολής οι buffers που είχαν χρησιμοποιηθεί για την λήψη των μηνυμάτων παρέχονται πάλι πίσω στο σύστημα. Το πλεονέκτημα αυτής της διαδικασίας είναι ότι αν πρόκειται πχ για εγγραφή αποφεύγουμε την ενδιάμεση αποθήκευση των δεδομένων σε κάποιον άλλον καταχωρητή. Το πρόβλημα είναι όμως ότι μια διαδικασία I/O είναι πολύ ποιο χρονοβόρα απ' ότι η λήψη κάποιου καινούργιου μηνύματος. Γι' αυτό και έχουμε φροντίσει να παρέχουμε όσους περισσότερους buffers μπορούμε στο σύστημα για να τοποθετούνται εκεί τα δεδομένα όσο θα εκτελείται κάποια διαδικασία I/O.

- **Λειτουργία Αποστολής** Η λειτουργία της αποστολής είναι απλή. Λαμβάνουμε τα έτοιμα μηνύματα προς αποστολή από την ουρά "εξόδου", με την προσοχή τα απεσταλμένα μηνύματα να μην ξεπεράσουν τα διαθέσιμα tokens. Η αποστολή γίνεται με την χρήση της συνάρτησης gm_send_with_callback(). Μετά την αποστολή του το μήνυμα τοποθετείται σε λίστα απ' όπου θα αφαιρεθεί με την ολοκλήρωση της αποστολής. Όπως έχουμε περιγράψει στην παράγραφο 5.2.5 ο μηχανισμός της αποστολής είναι ασύγχρονος. Για να διαπιστώσουμε ότι η διαδικασία έχει ολοκληρωθεί, θα πρέπει πάλι να κοιτάξουμε την ουρά γεγονότων της gm_port, αυτή τη φορά με την χρήση της συνάρτησης gm_receive(). Αν κάποια αποστολή έχει ολοκληρωθεί τότε θα λάβουμε ένα γεγονός τύπου GM_SENT_EVENT, το οποίο και θα περάσουμε στην συνάρτηση gm_unkown(). Η τελευταία με την σειρά της θα καλέσει την συνάρτηση "callback" που είχαμε καθορίσει κατά την αποστολή του μηνύματος, και η οποία θα σημαδεύσει ότι το μήνυμα ολοκληρώθηκε. Τέλος αφαιρούμε από την λίστα που αναφέραμε παραπάνω όσα μηνύματα έχει ολοκληρωθεί η αποστολή τους και τοποθετούμε τους buffers που είχαμε χρησιμοποιήσει πάλι πίσω στις αντίστοιχες ουρές, έτσι ώστε να χρησιμοποιηθούν εκ νέου.

  Θα πρέπει εδώ να επισημάνουμε την χρήση της συνάρτησης gm_receive() η οποία δεν μπλοκάρει την εκτέλεση του νήματος αποστολής, αλλά κάνει στην ουσία polling στην ουρά γεγονότων. Ακολουθούμε αυτή την επιλογή, γιατί θέλουμε το νήμα να μην περιμένει για κανένα άλλο γεγονός, παρά για την λήψη κάποιου καινούργιου μηνύματος από την ουρά

"εξόδου", γιατί υπάρχει ο κίνδυνος deadlock, αν το νήμα περιμένει για κάποιο καινούργιο γεγονός από το GM ενώ αυτό δεν υπάρχει.

5.5.1.1 Παρατηρήσεις

Θα πρέπει εδώ να παρατηρήσουμε ότι για να πραγματοποιηθεί κάποια διαδικασία I/O προς τον δίσκο, θα γίνει στην ουσία μια κλήση συστήματος write ή read, με ότι αυτό μπορεί να συνεπάγεται στην συνολική καθυστέρηση στην διεκπεραίωση της αίτησης και στον φόρτο που επιβάλλεται στον επεξεργαστή Στα παρακάτω άρθρα [11, 8] έχει γίνει μια προσπάθεια θα χρησιμοποιηθούν ο επεξεργαστής και οι μηχανές DMA της κάρτας Myrinet, για την απευθείας πραγματοποίηση της αίτησης I/O από την ίδια την κάρτα, και να παρακαμφθούν τα ενδιάμεσα επίπεδα του πυρήνα του Linux. Οι μετρήσεις που παρουσιάζονται φαίνεται να επαληθεύουν την αρχική υπόθεση ότι θα έχουμε αύξηση των επιδόσεων με παράλληλη μείωση της χρήσης του επεξεργαστή.

5.5.2 GM client

```
struct myri_client{
        struct gm_port  *send_port;
        struct gm_port  *rcv_port;
        gm_u32_t        myri_unit;
        uint32_t        sector_size;
        uint32_t        chunk_size;
        char            *server_name;
        gm_u32_t        server_node_id;
        gm_u32_t        receive_tokens;
        gm_u32_t        send_tokens;
        gm_u32_t        rcv_conf_tokens;
        gm_u32_t        rcv_header_tokens;
        gm_u32_t        rcv_buffer_tokens;
        struct list     *confs;
        struct list     *headers;
        struct list     *buffers;
        pthread_mutex_t mutex;
};
```

Στο σχήμα 3.6 είδαμε την δομή που θα πρέπει να έχει το υποσύστημα του vRAID που για να υλοποιήσει την επικοινωνία με την χρήση του GM API. Ο κώδικας που προηγείται περιέχει όλες τις βασικές δομές που χρειάζονται για την λειτουργία του.

Το μέρος που υλοποιήσαμε είναι αυτό του GM Client Core, γιατί η υλοποίηση και των επιμέρους clients απαιτεί αλλαγές στις δομές και στον τρόπο

λειτουργίας του vRAID controller. Όπως περιγράψαμε παραπάνω, ο αριθμός των διαθέσιμων tokens που έχουμε προς αποστολή είναι περιορισμένος, και θα χρειαστεί να επιβάλλουμε κάποιο είδος ελέγχου ροής στα εισερχόμενα μηνύματα προς αποστολή. Αυτό μπορεί να γίνει μόνο με την ύπαρξη μόνο μιας ουράς αποστολής, που θα ανήκει στο Core, και την οποία θα μπορεί να σταματά ή να ξεκινά ανάλογα με τα διαθέσιμα tokens. Ως συνέπεια των παραπάνω θα πρέπει οι μεμονωμένες ουρές που έχει κάθε συσκευή στην δομή vraid_device (4.3), να αντικατασταθούν από την ενιαία ουρά του Core.

Επίσης μέσα στον Core θα πρέπει να προστεθεί μια δομή που αντιστοιχήσει τις ουρές pending με τους αντίστοιχους απομακρυσμένους δίσκους. Έτσι όταν ένα μήνυμα φτάνει, τότε θα αφαιρείται η αίτηση από την αντίστοιχη ουρά pending, και το αποτέλεσμα θα μεταβιβάζεται στα ανώτερα επίπεδα τους vRAID με την χρήση της συνάρτησης vraid_connection_enqueue().

Κατά τ' άλλα ο τρόπος λειτουργίας του Core είναι όμοιος με αυτόν του GM server. Δηλαδή έχουμε δεσμεύσει εκ των προτέρων τους καταχωρητές που αντιστοιχούν σε ισάριθμα tokens αποστολής και λήψης. Έχουμε δύο νήματα, ένα για την λήψη και ένα για την αποστολή μηνυμάτων. Λαμβάνουμε τις εισερχόμενες αιτήσεις από την ουρά εισόδου, παράγουμε τα μηνύματα προς αποστολή που μπαίνουν σε λίστα αναμονής, και τα αποστέλλουμε όταν έχουμε διαθέσιμα tokens. Στο νήμα λήψης με την άφιξη κάποιου καινούργιου μηνύματος, βρίσκουμε την αίτηση που βρίσκεται στην pending ουρά, και προωθούμε το αποτέλεσμα στα υψηλότερα layers του vRAID.

Κεφάλαιο 6

Μετρήσεις

6.1 Χαρακτηριστικά μετρήσεων

Οι μετρήσεις μας πραγματοποιήθηκαν σε δύο μηχανήματα που μου παραχώρησε το εργαστήριο. Η σύνδεσή τους είναι αυτή που φαίνεται στον πίνακα 6.1.

Για την πραγματοποίηση των μετρήσεων χρησιμοποιήθηκε ένα partition μεγέθους περίπου 1 GigaByte, χωρίς την ύπαρξη κάποιου συστήματος αρχείων από πάνω, όπου και γράφονταν ή διαβάζονταν raw δεδομένα. Η απουσία του συστήματος αρχείων επιλέχτηκε έτσι ώστε να μην χρησιμοποιήσουμε τους μηχανισμούς βελτιστοποίησης που παρέχουν και επιταχύνουν σειριακές αναγνώσεις ή εγγραφές. Επίσης προσπαθήσαμε να παρακάμψουμε τις διάφορες caches και τους μηχανισμούς ασύγχρονης υλοποίησης του εντολών I/O. Τα παρακάνω στοιχεία δεν είναι ευπρόσδεκτα στο σύστημά μας, γιατί μπορεί να προκαλέσουν εν αγνοία μας απώλεια δεδομένων στην περίπτωση που κάποιο σύστημα σταματήσει απρόσμενα να λειτουργεί (λόγω πχ διακοπής ρεύματος). Επίσης θέλουμε να δούμε τον συνδυασμό σκληρός δίσκος - δίκτυο επικοινωνίας πως συμπεριφέρεται ενιαία και κατά προτίμηση χωρίς την διαμεσολάβηση κάποιου λειτουργικού συστήματος. Γι' αυτό το λόγο επιβάλλαμε οι εντολές I/O να γίνονται σύγχρονα, περνώντας ως παράμετρο κατά το άνοιγμα του block device που αντιστοιχεί στο parition την παράμετρο O_SYNC. Μια καλύτερη προσέγγιση μπορεί να ήταν η χρήση της παραμέτρου O_DIRECT που θεωρητικά παρακάμπτει της caches του λειτουργικού, όμως η χρήση της δεν μας διευκόλυνε ούτε προγραμματιστικά αλλά και οι αρχικές μετρήσεις που πήραμε δεν είχαν τα φυσιολογικά αποτελέσματα που περιμέναμε.

Οι μετρήσεις μας περιλαμβάνουν δύο διαφορετικές διεπαφές επικοινωνίας. Η πρώτη είναι τα πρωτόκολλα που χρησιμοποιούνται για την επικοινωνία του vRAID με τα απομακρυσμένα συστήματα επικοινωνίας. Η δεύτερη αφορά συ-

Λειτουργικό	Linux 2.6.16.17
Επεξεργαστής	2 * Intel(R) Xeon(TM) CPU 2.80GHz cache size 1024 KB
Μνήμη	2076336 kB
Δίσκος	Maxtor 6Y080M0 SATA 1 80 Giga
Κάρτα Ethernet	2 * Intel Corporation 82546GB Gigabit Ethernet Controller
Κάρτα Myrinet	Myrinet 2000 Scalable Cluster Interconnect LANai10.0 2040K bytes

Πίνακας 6.1: Η σύνθεση των μηχανημάτων στα οποία έγιναν οι μετρήσεις.

στήματα που μπορούσε να επικοινωνήσουμε μέσω εντολών I/O σε αντίστοιχες συσκευές block του Linux, όπως ο σκληρός δίσκος και η block device του AoE. Και στις δυο περιπτώσεις μετρήσαμε το bandwidht και την καθυστέρηση (latency) του συστήματος για διαφορετικά μεγέθη αιτήσεων, τόσο για σειριακές όσο και τα τυχαίες προσπελάσεις του δίσκου .

Οι σειριακές προσπελάσεις έγιναν σε όλο με μήκος του απομακρυσμένου δίσκου (partition στην περίπτωσή μας) . Οι τυχαίες προσπελάσεις έγιναν για σταθερό αριθμό αιτήσεων στο σύνολό τους. Η κατανομή των τυχαίων προσπελάσεων ήταν κανονική με μέση τιμή το μέσο του δίσκου και διασπορά το μισό του μεγέθους του.

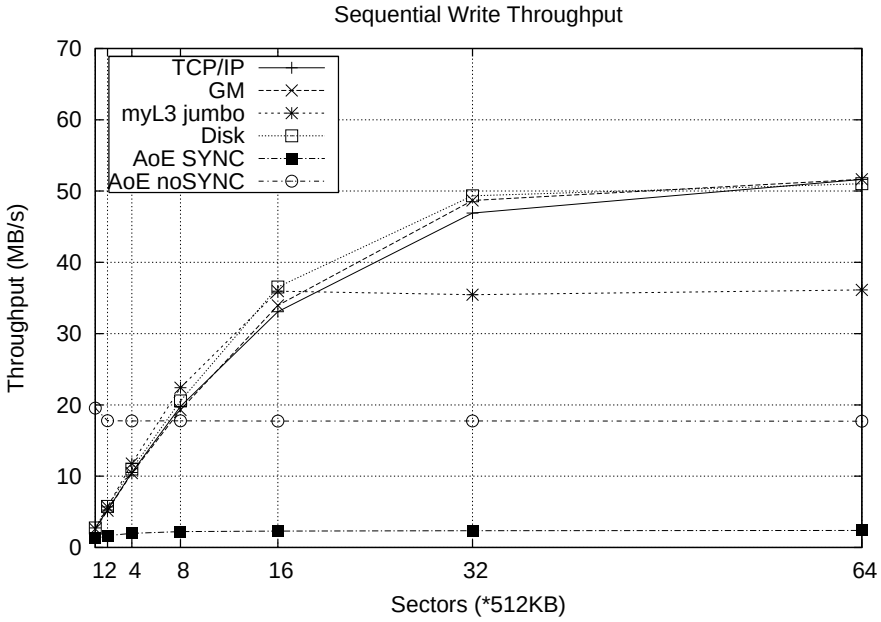
Για την περίπτωση των πρωτοκόλλων επικοινωνίας του VRAID. για να μετρήσουμε το throughput, εισάγαμε στην ουρά εισόδου της συσκευής ένα δεδομένο αριθμό αιτήσεων και στην συνέχεια περιμέναμε να αφαιρέσουμε όλες τις αναμενόμενες απαντήσεις. Η επικοινωνία ήταν ασύγχρονη από την μεριά του vRAID (οι εγγραφές στον απομακρυσμένο δίσκο γίνονται πάντα σύγχρονα), με σκοπό να μπορέσουμε να οδηγήσουμε το σύστημα στα όριά του. Γνωρίζοντας τον χρόνο έναρξης, τον χρόνο λήψης και το συνολικό μέγεθος των αιτήσεων που στείλαμε, υπολογίσαμε το throughput. Για την μέτρηση της καθυστέρησης εισάγαμε πάλι αιτήσεις μόνο που αυτή την φορά η επικοινωνία στην μεριά του vRAID ήταν σύγχρονη, δηλαδή μετά την αποστολή μιας αίτησης περιμέναμε μέχρι να λάβουμε την απάντηση. Η διαφορά χρόνου από την στιγμή της εισαγωγής της αίτησης μέχρι και την λήψη της απάντησης αποτελεί την καθυστέρηση του συστήματος.

Για την περίπτωση των block devices ακολουθήσαμε την ίδια λογική με

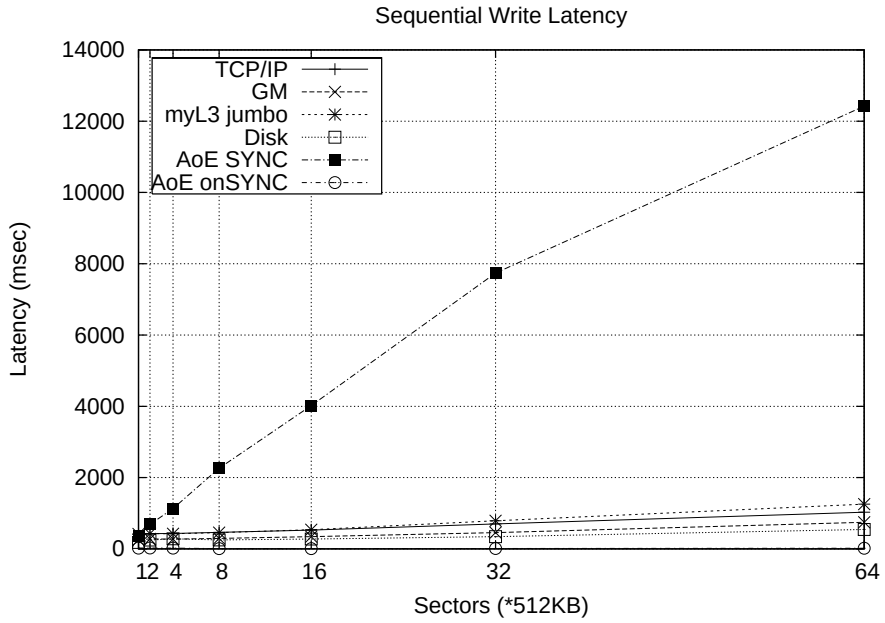
παραπάνω. Οι μετρήσεις που πήραμε για τον δίσκο έγιναν με την χρήση της παραμέτρου O_SYNC. Στην περίπτωση του AoE έγιναν δύο τύποι μετρήσεων, ένας όπου η επικοινωνία τόσο στον απομακρυσμένο δίσκο όσο και στο block device του AoE είναι σύγχρονη, και ένας όπου είναι ασύγχρονη.

Οι μετρήσεις για το TCP/IP και για το AoE έγιναν για Etherent MTU 1500. Οι μετρήσεις για το myL3 έγιναν για Ethernet MTU 9000.

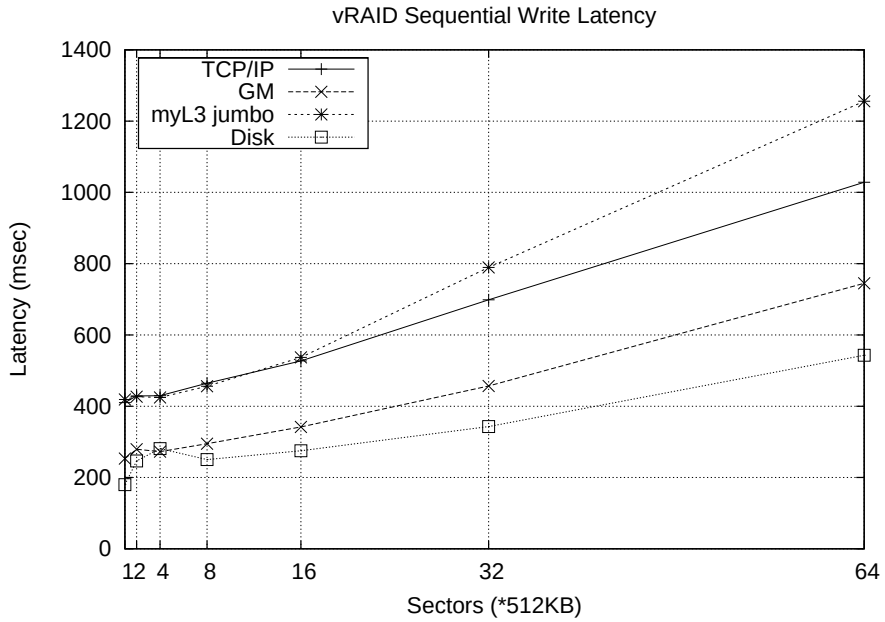
6.2 Αποτελέσματα



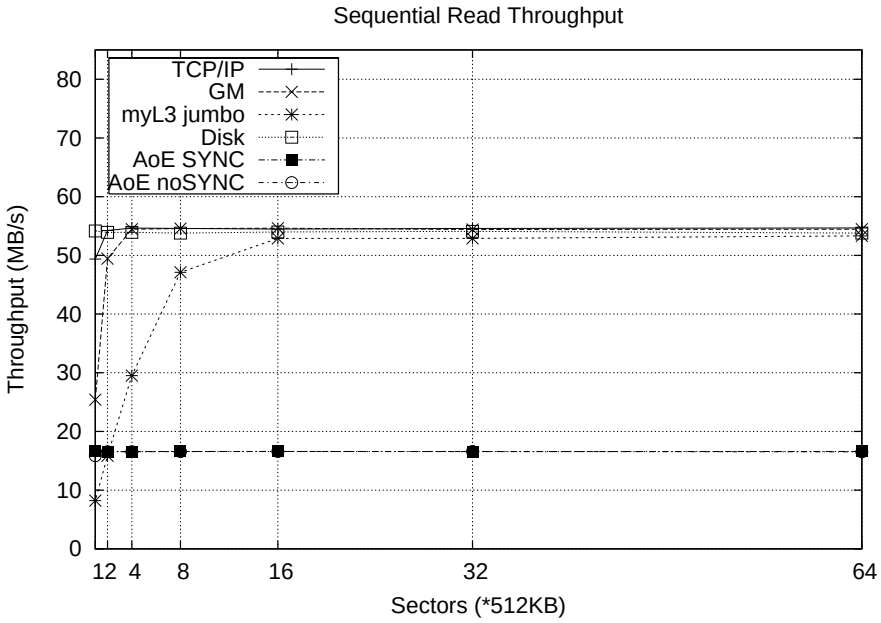
Σχήμα 6.1



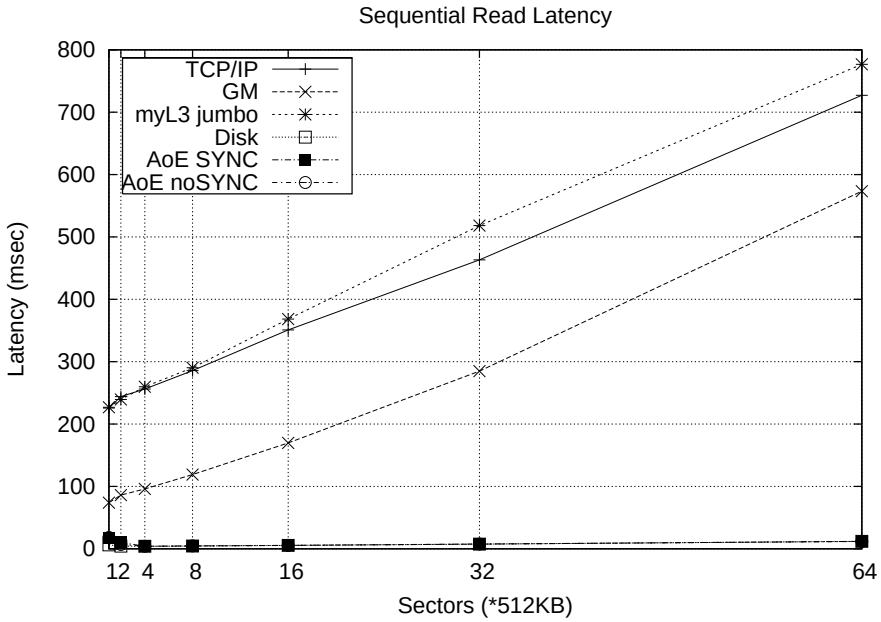
Σχήμα 6.2



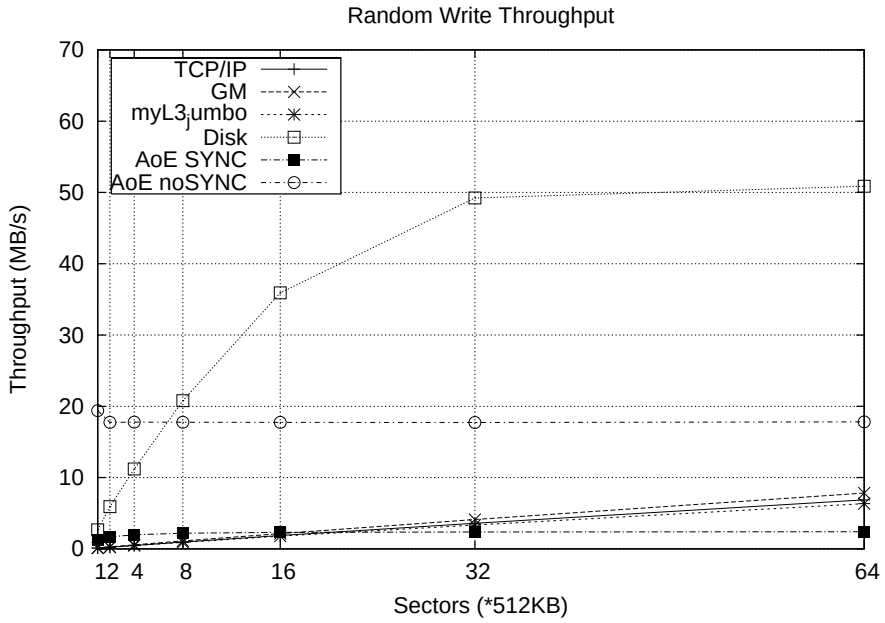
Σχήμα 6.3



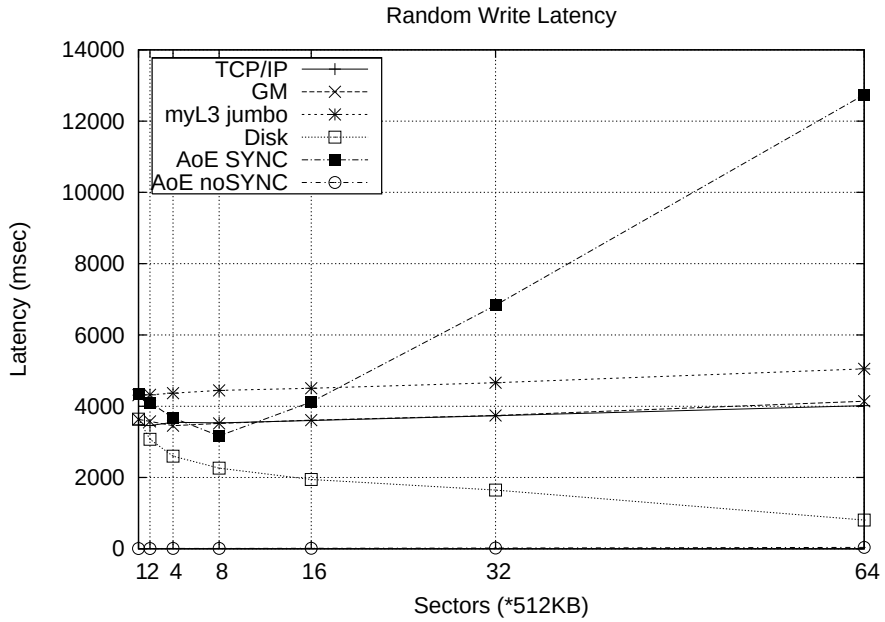
Σχήμα 6.4



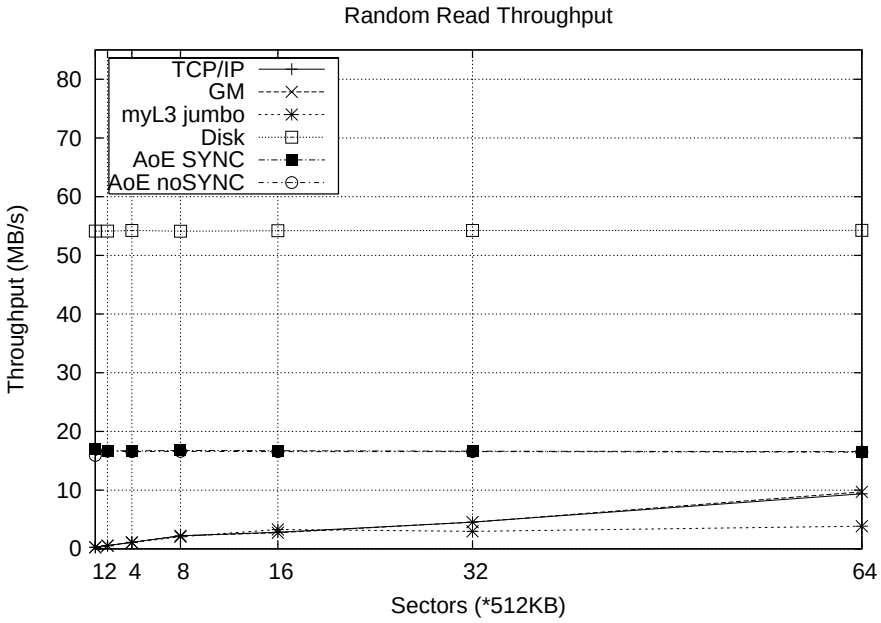
Σχήμα 6.5



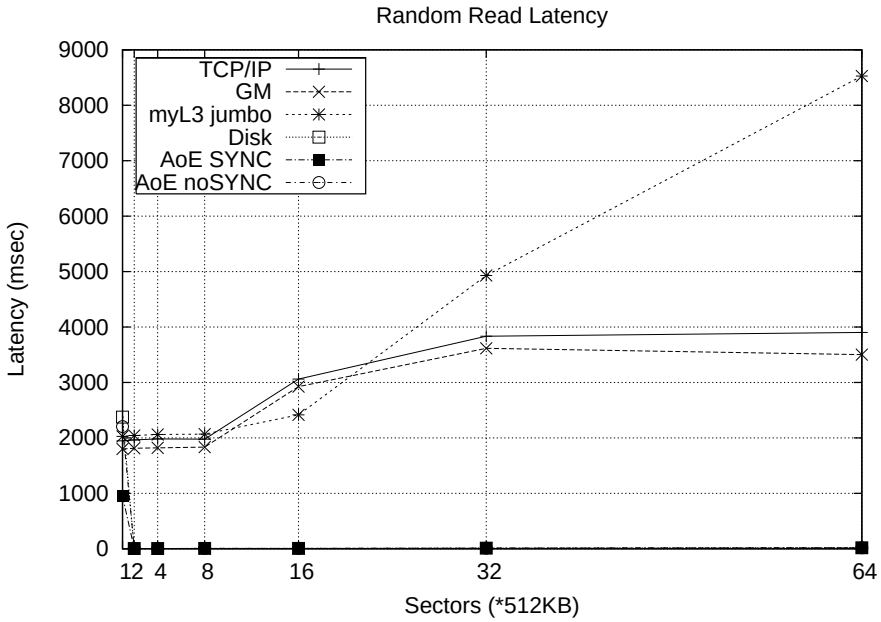
Σχήμα 6.6



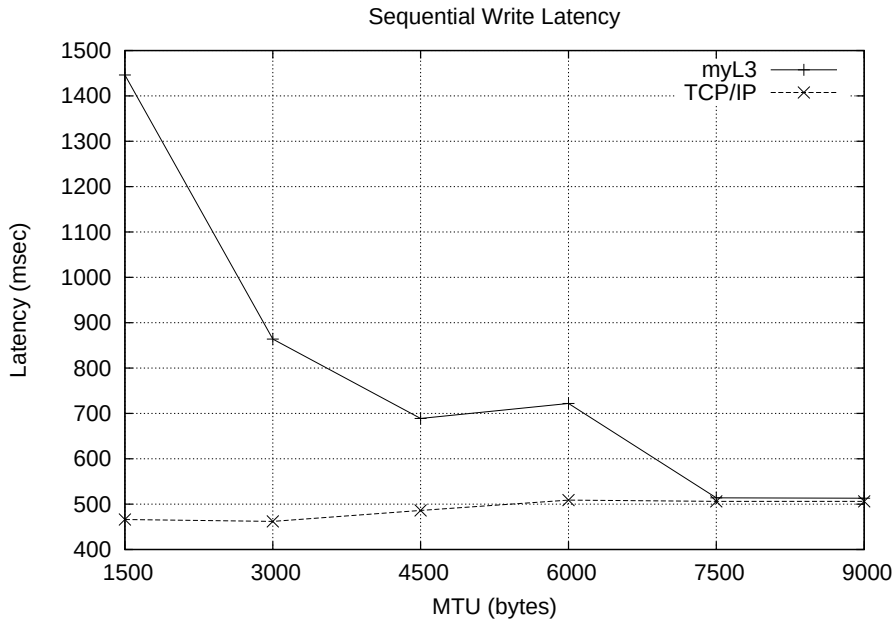
Σχήμα 6.7



Σχήμα 6.8



Σχήμα 6.9



Σχήμα 6.10: Οι παραπάνω μετρήσεις έγιναν για μηνύματα μήκους 12 sectors. Η τιμή της καθυστέρησης μιας σύγχρονης εγγραφής στο δίσκο είναι περίπου 250 msec

6.2.1 Σχόλια

Στα παραπάνω διαγράμμα παρουσιάζουμε μαζί τόσο τις μετρήσεις που έγιναν για την ομάδα των υποσυστήματα του vRAID όσο και αυτές που έγιναν για την ομάδα των block devices, για λόγους οικονομίας. Τα αποτελέσματα μεταξύ διαφορετικών ομάδων δεν είναι άμεσα συγκρίσιμα, αλλά μπορεί να γίνει μεταξύ τους μόνο ποιοτική αντιπαραβολή των αποτελεσμάτων.

Κατά την πραγματοποίηση των μετρήσεών μας προσπαθήσαμε να απενεργοποιήσουμε κατά το δυνατόν τους μηχανισμούς του Linux που επιταχύνουν τις εντολές I/O, γι' αυτό και χρησιμοποιήσαμε την παράμετρο O_SYNC κατά το άνοιγμα των αρχείων. Τελικά όπως φαίνεται και από τις μετρήσεις καταφέραμε να επηρεάσουμε μόνο τις εγγραφές. Αντίθετα στον μηχανισμό των αναγνώσεων παρεμβάλλεται το λειτουργικό πραγματοποιώντας βελτιώσεις όπως ομαδοποίηση των I/O, προανάκληση (prefetcing) sectors ή ακόμα και caching. Αυτό μπορεί να φανεί καθαρά στα σχήματα 6.4 και 6.8 όπου μπορούμε να δούμε ότι οι επιδόσεις του δίσκου κυμαίνονται σε υψηλά επίπεδα είτε πρόκειται για συνεχόμενες είτε για τυχαίες αναγνώσεις. Με την ίδια λογική μπορούν να εξηγηθούν και οι επιδόσεις του AoE που εμφανίζονται σταθερές ανεξαρτήτως του μεγέθους της εντολής I/O που εκτελείται, όσο και η ελάχιστη καθυστέρηση που παρουσιάζει.

Κεφάλαιο 7

Επίλογος-Συμπεράσματα

7.1 Αξιολόγηση Αποτελεσμάτων

Η αρχική μας ελπίδα ότι θα μπορούσε η απευθείας χρήση του Ethernet να αποδειχθεί αποδοτική και ανταγωνιστική υπαρχουσών λύσεων δυστυχώς διαψεύστηκε, το μόνο του πλεονέκτημα βέβαια παραμένει μόνο το χαμηλό του κόστος και η ευρεία χρήση του. Ας δούμε όμως αναλυτικά τα συμπεράσματα των μετρήσεών μας.

Η βασική αρχή που διαπερνά την επικοινωνία I/O μ' έναν σκληρό δίσκο είναι ότι οι αιτήσεις I/O πρέπει να είναι όσο το δυνατόν μεγαλύτερες γίνεται. Αυτή είναι και η βασική αιτία που στην ουσία προσδιορίζει την επίδοση των διαφόρων υλοποιήσεών μας. Στο σχήμα 6.1 μπορούμε να διαπιστώσουμε την αύξηση των επιδόσεων με την αύξηση των προς εγγραφή δεδομένων. Παραμένει λοιπόν ο δίσκος να αποτελεί την στενωπό του συστήματός μας, με το δίκτυο να είναι κατά μεγάλο μέρος του ανεκμετάλλευτο. Σημαντική θα είναι η συμβολή του δικτύου στην μείωση της καθυστέρησης, καθώς ο χρόνος μετάδοσης μεγάλων μηνυμάτων δεν είναι αμελητέος. Άμμεση συνέπεια λοιπόν να μαντέψουμε ότι νικητής θα βγεί η εφαρμογή αυτή που θα πραγματοποιεί τις λιγότερες κλήσεις I/O με το μεγαλύτερο δυνατό μέγεθος, και θα έχει την μικρότερη καθυστέρηση στην μετάδοση του μηνύματος.

Νικητής λοιπόν ανακηρύσσεται η υλοποίηση που είναι βασισμένη σε GM-Myrinet, βάσει των σχημάτων 6.1 και 6.3. Δυστυχώς τα σχήματα που αναφέρονται σε τυχαίες προσπελάσεις δεν μπορούν να μας βοηθήσουν, γιατί η συνεισφορά του δίσκου στην καθυστέρηση είναι πολύ μεγάλη και η διαφορά των επιδόσεων του GM και του TCP/IP προσεγγίζει το στατιστικό σφάλμα. Η αιτία που το GM έχει τόσο καλές επιδόσεις είναι γιατί πληρεί τις προϋποθέσεις που περιγράψαμε παραπάνω. Μπορεί να προσφέρει δηλαδή μεγάλα μηνύματα προς εγγραφή και η καθυστέρηση που εισάγει στο επίπεδο του δικτύου είναι

πολύ μικρότερη σε σχέση με αυτή του TCP/IP-Ethernet.

Δεύτερη σε επιδόσεις αναδεικνύεται η υλοποίηση σε TCP/IP. Η βασική αιτία είναι ότι και αυτό με την σειρά του μας επιτρέπει να πραγματοποιούμε εντολές I/O σε μεγάλο εύρος δεδομένων.

Ερχόμαστε τώρα λοιπόν στην δυσχερή θέση να περιγράψουμε γιατί τα πρωτόκολλα myL3 και AoE δεν κατάφεραν να εκπληρώσουν τις προσδοκίες μας. Οι αιτίες είναι οι παρακάτω:

- **Μορφή πρωτοκόλλου:** Όλα τα πρωτόκολλα είναι της μορφής αποστολή αίτησης - πραγματοποίηση I/O - αναμονή απάντησης. Αυτή η προσέγγιση αν και απλή, εισάγει περιορισμούς στις επιδόσεις αφού δεν επιτρέπει την ομαδοποίηση των εντολών I/O, και επιβάλλει την άμμεση εκτέλεσή τους για την αποστολή της απάντησης.

- **MTU:** Αποτελεί στην ουσία τον ποιο σημαντικό παράγοντα που μειώνει τις επιδόσεις, αφού στην ουσία περιορίζει τα δεδομένα που μπορούν να μεταφέρουν, άρα και το μέγεθος του I/O που μπορεί να εκτελεστεί. Έχουμε λοιπόν περισσότερα και μικρότερα I/O σε σύγκριση με το GM ή το TCP/IP με αποτέλεσμα να έχουμε και μείωση των επιδόσεων. Στο σχήμα 6.10 μπορούμε να δούμε εμφανώς πώς μειώνεται η καθυστέρηση με παράλληλη αύξηση του MTU. Ενώ η επίδοση του TCP/IP παραμένει πρακτικά σταθερή, η καθυστέρηση για το myL3 μειώνεται, στην ουσία υποτριπλασιάζεται, όσα και τα πακέτα που χρειάζεται να στείλει, όσα και τα I/Os που πραγματοποιεί.

- **Μέγεθος παραθύρου - Ουρά εκκρεμών αιτήσεων:** Και τα δύο πρωτόκολλα έχουν σταθερό μέγεθος παραθύρου που έχει προσδιοριστεί δοκιμαστικά. Έτσι δεν είναι σε θέση να εκμεταλλευτούν ρυθμίσεις του συστήματος με μεγαλύτερους buffers αποστολής και λήψης που στην περίπτωσή τους θα επιτάχυνε την λειτουργία τους.

- **Μη ασφαλές μέσο επικοινωνίας:** Το σημαντικό πρόβλημα είναι ότι το Ethernet δεν είναι ασφαλές μέσο επικοινωνίας. Δεν δίνει άμεσα την δυνατότητα στον αποστολέα αν διαπιστώσει την επιτυχία της αποστολής όπως μας επιτρέπει το GM. Αναγκαζόμαστε έτσι να λειτουργούμε μέσω επιβεβαιώσεων που εισάγει καθυστέρηση. Το σημαντικότερο όμως είναι ότι η επαναποστολή ενός μηνύματος αποφασίζεται από την αποτυχία λήψης απάντησης. Έτσι είναι πιθανό να έχουμε απώλεια του πακέτου κατά την αποστολή οπότε η επανάληψή της είναι απόλυτα θεμιτή. Η απώλεια όμως της απάντησης εισάγει επιπλέον επεξεργαστικό φορτίο που είναι άχρηστο. Το τελευταίο στοιχείο καθιστά τέτοια πρωτόκολλα μη ιδανικά για την υλοποίηση μηχανισμών κλειδωμάτων καθώς η απώλεια της απάντησης μπορεί να οδηγήσει σε επανάληψη και πιθανό deadlock.

Επίσης στο σχήμα 6.10 μπορούμε να δούμε ότι το myL3 δεν καταφέρνει ποτέ στην ουσία να εμφανίσει καθυστέρηση μικρότερη από αυτή του TCP/IP, αλλά τουλάχιστον επιτυγχάνει περίπου την ίδια. Η αιτία είναι ότι πρόκειται για userspace εφαρμογή και η διεπαφή επικοινωνίας είναι το PF_PACKET socket, γεγονός που σημαίνει ότι δε μπορούμε ν' αποφύγουμε αντιγραφές μνήμης ανάμεσα στο userspace και στο kernelspace. Πιθανώς η υλοποίησή του μέσα στον πυρήνα να εμφάνιζε χαμηλότερη καθυστέρηση. Σε κάθε περίπτωση το κέρδος της παράκαμψης του TCP/IP stack δεν καταφέραμε να είναι τόσο σημαντικό ώστε να σημάνει και πραγματικό κέρδος στην επικοινωνία. Πιθανώς σε σύγκριση με το iSCSI τα αποτελέσματα να ήταν καλύτερα.

Το τελικό συμπέρασμα είναι ότι τα πρωτόκολλα που υλοποιούνται ακριβώς πάνω από το Ethernet υπολείπονται σε επιδόσεις των υπολοίπων. Ο δίσκος εισάγει στην ουσία όλη την καθυστέρηση στο σύστημα, οπότε το πιθανό κέρδος από την παράληψη του TCP/IP να είναι αμελητέο. Το μόνο δυνατό στοιχείο τους είναι ότι εκμεταλλεύονται την υπάρχουσα υποδομή. Για να είναι πραγματικά συμφέρουσα η χρήση τους σε συστήματα αποθήκευσης θα πρέπει να δημιουργηθούν διεπαφές επικοινωνίας σαν αυτή του EtherBlade της Coraid, με πολύ χαμηλότερη τιμή, που να κάνει χρήση οπωσδήποτε των Jumbo Frames του Gigabit Ethernet για την επίτευξη καλύτερων επιδόσεων.

7.2 Μελλοντική Έρευνα

Η μελλοντική έρευνα όσον αφορά την οικογένεια πρωτοκόλλων που ανήκουν το AoE και το myL3 θα πρέπει αρχικά να επικεντρωθεί στην προσπάθεια ομαδοποίησης των εντολών I/O στην μεριά του server. Κατά την προσπάθεια ανάπτυξης του myL3 server έγινε προσπάθεια να ομαδοποιούνται τα I/O και να ανασχηματίζονται οι αρχικές αιτήσεις του vRAID. Κάτι τέτοιο στάθηκε όμως δύσκολο εξαιτίας της ανασφάλειας του μέσου. Η πιθανή η απώλεια πακέτων τόσο των αιτήσεων όσο και απαντήσεων εισήγαγε επιπλέον πολυπλοκότητα και στα δύο άκρα της επικοινωνίας που την καθιστούσε ασύμφορη. Επίσης το κόστος σε χρόνο κάποιας απώλειας αυξανόταν. Η εισαγωγή δυναμικού παραθύρου θα αύξανε τις επιδόσεις, με βασικό πρόβλημα όμως να είναι ότι η μονάδα μεταφερόμενης πληροφορίας είναι το μέγεθος ενός sector, που δυσχεραίνει την προσπάθεια.

Σε κάθε περίπτωση η λύση θα πρέπει να είναι απλή και αποτελεσματική. Δεν έχει νόημα τελικά να φτάσουμε να γράψουμε τελικά ξανά το TCP/IP. Μια λύση καλύτερη των υφισταμένων, μη βέλτιστη, και πολύ χαμηλού κόστους μπορεί όντως να χρησιμοποιηθεί για την δημιουργία πραγματικά Inexpencive (φτηνών) δικτυακών συστοιχιών RAID.

Τα μελλοντικά πρωτόκολλα επικοινωνίας υψηλών επιδόσεων με δίσκους θα

πρέπει να προσφέρουν τα πλεονεκτήματα του Myrinet. Ασφαλής επικοινωνία, χαμηλής καθυστέρησης, όπου το MTU να μην επιβάλλει την πραγματοποίηση μικρών σε μέγεθος εντολών I/O. Οποιαδήποτε υλοποίηση παραβλέψει την συμπεριφορά του δίσκου δεν θα καταφέρει πραγματική αύξηση των επιδόσεων.

Βιβλιογραφία

[1] *Information Techonology - AT Attachement with Packet Interface - 6 (ATA/ATAPI-6).*

[2] H. Peter Anvin. The mathematics of raid-6. December 2004.

[3] M. Blaum, J. Brady, J. Bruck, and J. Menon. Evenodd: an optimal scheme for tolerating double disk failures in raid architectures. In *ISCA '94: Proceedings of the 21ST annual international symposium on Computer architecture*, pages 245–254, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[4] Mario Blaum and Ron M. Roth. On lowest density MDS codes. *IEEE Transactions on Information Theory*, 45(1):46–59, 1999.

[5] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, 1995.

[6] Antony Chazapis, Giorgos Verigakis, and Nectarios Koziris. Contructing virtual sans with vraid: A technical report. 2006.

[7] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, 1994.

[8] Olivier Cozette, Cyril Randriamaro, and Gil Utard. Read2: Put disks at network level. *ccgrid*, 00:698, 2003.

[9] Robert H. Deng, Feng Bao, and Jia-Chen Shen. New efficient mds array codes for raid part ii: Rabin-like codes for tolerating multiple (greater than or equal to 4) disk failures. *IEEE Trans. Comput.*, 54(12):1473–1483, 2005. Senior Member-Gui-Liang Feng.

[10] Gui-Liang Feng, Robert H. Deng, Feng Bao, and Jia-Chen Shen. New efficient mds array codes for raid part i: Reed-solomon-like codes for tolerating three disk failures. *IEEE Transactions on Computers*, 54(9):1071–1080, 2005.

[11] Patrick Geoffray. Opiom: Off-processor io with myrinet. *ccgrid*, 00:261, 2001.

[12] Garth A. Gibson and Rodney Van Meter. Network attached storage architecture. *Commun. ACM*, 43(11):37–45, 2000.

[13] Lisa Hellerstein, Garth A. Gibson, Richard M. Karp, Randy H. Katz, and David A. Patterson. Coding techniques for handling failures in large disk arrays. *Algorithmica*, 12(2/3):182–208, 1994.

[14] Ping-Hsun Hsieh, Ing-Yi Chen, Yu-Ting Lin, and Sy-Yen Kuo. An xor based reed-solomon algorithm for advanced raid systems. *dft*, 00:165–172, 2004.

[15] Myricom Inc. Guide to myrinet-2000 switches and switch networks. Technical report, 2001.

[16] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988.

[17] Randy H. Katz, Garth A. Gibson, and David A. Patterson. Disk system architectures for high performance computing. Technical report, Berkeley, CA, USA, 1989.

[18] Edward K. Lee and Randy H. Katz. The performance of parity placements in disk arrays. In Hai Jin, Toni Cortes, and Rajkumar Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, pages 35–54. IEEE Computer Society Press and Wiley, New York, NY, 2001.

[19] S. Michael. Autonet: A high-speed, self-configuring local area network using pointto -point links, 1991.

[20] VITA Standards Organization. Myrinet-on-vme protocol specification draft standard. Technical report, 1998.

[21] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 109–116, New York, NY, USA, 1988. ACM Press.

[22] Coraid:: The Linux Storage People.

[23] Coraid:: The Linux Storage People. *AoE Description.*

[24] Coraid:: The Linux Storage People. *AoE Protocol.*

[25] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[26] J. S. Plank. Erasure codes for storage applications. Tutorial Slides, presented at *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, `http://www.cs.utk.edu/~plank/plank/papers/FAST-2005.html`, 2005.

[27] J. S. Plank and Y. Ding. Note: Correction to the 1997 tutorial on reed-solomon coding. Technical Report CS-03-504, University of Tennessee, April 2003.

[28] J. S. Plank and L. Xu. Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. In *NCA-06: 5th IEEE International Symposium on Network Computing Applications*, Cambridge, MA, July 2006.

[29] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. Unix network programming volume 1, third edition: The sockets networking api. 2003.

[30] Lihao Xu and Jehoshua Bruck. X-code: MDS array codes with optimal encoding. *IEEE Transactions on Information Theory*, 45(1):272–276, 1999.