ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αξιολόγηση Επιδόσεων Χρονοδρομολογητών σε Συστοιχίες Υπολογιστών

Παναγιώτα Α. Βυργιώτη

Νικόλαος Ε. Κοτζαλάς

Επιβλέποντες: Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ
Βασίλειος Τσέτσος, Υποψήφιος Διδάκτωρ ΕΚΠΑ

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2007

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αξιολόγηση Επιδόσεων Χρονοδρομολογητών σε Συστοιχίες Υπολογιστών

Παναγιώτα Α. Βυργιώτη

Α.Μ.: 1115200000014

Νικόλαος Ε. Κοτζαλάς

Α.Μ.: 1115200100190

ΕΠΙΒΛΕΠΟΝΤΕΣ:

Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ
Βασίλειος Τσέτσος, Υποψήφιος Διδάκτωρ ΕΚΠΑ

ΠΕΡΙΛΗΨΗ

Ο χρονοπρογραμματισμός είναι μια διαδικασία ανάθεσης διεργασιών σε ένα σύνολο υπολογιστικών πόρων. Είναι μια έννοια με κυρίαρχη θέση σχεδόν σε όλους τους τομείς της Πληροφορικής. Η παρούσα μελέτη περιστρέφεται γύρω ακριβώς από αυτή τη έννοια. Οι λύσεις χρονοπρογραμματισμού που προσφέρονται, τόσο σε ερευνητικό όσο και σε εμπορικό επίπεδο, παρέχουν μια πληθώρα δυνατοτήτων όσον αφορά τον ευέλικτο χρονοπρογραμματισμό διεργασιών. Ωστόσο, οι ολοένα αυξανόμενες απαιτήσεις για αξιοπιστία, υψηλή απόδοση και παραγωγικότητα, μετατρέπουν το χώρο του χρονοπρογραμματισμού σε πρόσφορο έδαφος για την ανάπτυξη λύσεων με επιπρόσθετη λειτουργικότητα. Σε αυτή την εργασία παρουσιάζονται δυο λύσεις χρονοπρογραμματισμού εργασιών σε κατανεμημένο περιβάλλον, με κοινά χαρακτηριστικά την υψηλή απόδοση και την ανεξαρτησία από λειτουργικά συστήματα και λοιπές τεχνολογίες σε κατώτερα επίπεδα. Συγκεκριμένα, γίνεται μια εκτενής αναφορά στην γενική σχεδίαση και αρχιτεκτονική και των δυο συστημάτων χρονοπρογραμματισμού εργασιών. Η όλη αναφορά στις δυο περιγραφείσες λύσεις χρονοπρογραμματισμού ολοκληρώνεται με μια ανάλυση των επιδόσεων τους και μια συγκριτική μελέτη η οποία βασίζεται τόσο σε ρεαλιστικά όσο και σε πιο απαιτητικά σενάρια αξιολόγησης.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: χρονοπρογραμματισμός εργασιών

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: χρονοπρογραμματισμός εργασιών, εργασία, ρυθμαπόδοση, συστοιχία, κλιμάκωση

ABSTRACT

Scheduling can be defined as the process of assigning tasks to a set of resources. Scheduling is a dominant term in many fields in the area of computing, such as operating systems, computer networks and enterprise platforms. The scheduling solutions that are offered, both commercial and scientific, provide a wide variety of possibilities. However, the ever increasing demands for reliability, high performance and productivity are making the scheduling area a promising one for developing solutions with more complex functionality. In this thesis, two scheduling solutions are presented. Both systems seem to meet the requirements even of the most demanding applications, since they are capable of working in distributed environments and exhibit high performance and independence from the operating system. In particular, the first part presents the architecture of both frameworks, whereas at the second part a comparison between these scheduling solutions is attempted by testing them under realistic and more stressful artificial scenarios.

SUBJECT AREA: job scheduling

KEY WORDS: job scheduling, task, throughput, cluster, scalability

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ

Η παρούσα εργασία πραγματοποιήθηκε στα πλαίσια εκπόνησης Πτυχιακής εργασίας στο πλαίσιο του προγράμματος Προπτυχιακών Σπουδών του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Αθηνών, υπό την επίβλεψη του καθηγητή Κου Χατζηευθυμιάδη Ευστάθιου.

Θα θέλαμε να ευχαριστήσουμε θερμά τον υποψήφιο διδάκτορα Κο Τσέτσο Βασίλειο για την πολύτιμη βοήθεια, τις εύστοχες παρατηρήσεις του και το χρόνο που διέθεσε καθώς και τον επιβλέποντα καθηγητή κο Χατζηευθυμιάδη Ευστάθιο, για την καθοδήγησή του και την εποικοδομητική συνεργασία μας κατά την διάρκεια εκπόνησης της εργασίας αυτής.

.

Οκτώβριος 2007

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

1.1 Γενικά

Σε γενικές γραμμές, ο χρονοπρογραμματισμός μπορεί να διαχωριστεί σε δυο κατηγορίες:

- Low Level Scheduling
- Application Level – Enterprise Scheduling.

Η πρώτη κατηγορία περιλαμβάνει τεχνικές και συστήματα scheduling σε multitasking λειτουργικά συστήματα και στοιχεία δικτύου. Αυτά τα συστήματα πρέπει να είναι υπερβολικά γρήγορα και να υποστηρίζουν τεχνικές χρονοπρογραμματισμού βασισμένες στην προτεραιότητα της κάθε εργασίας. Σε αντίθεση, η δεύτερη κατηγορία περιλαμβάνει χρονοπρογραμματιστές, τεχνικές εκτέλεσης διαδικασιών είτε σε πραγματικό χρόνο είτε σε κάποια μελλοντική προκαθορισμένη χρονική στιγμή, περιοδικά ή μη. Τέτοιοι χρονοπρογραμματιστές μπορεί να είναι υπεύθυνοι για την εκτέλεση μιας ομάδας εργασιών (batch jobs) ή για τη παροχή χρονικά εξαρτώμενων (time-triggered) υπηρεσιών. Τα κύρια χαρακτηριστικά τους οφείλουν να είναι η αξιοπιστία και η υψηλή κλιμακωσιμότητα (scalability) ώστε να αξιοποιηθούν στον τομέα των enterprise εφαρμογών.

Ένας enterprise χρονοπρογραμματιστής, σαν μέρος μιας ευρύτερης enterprise εφαρμογής, πρέπει να ικανοποιεί αρκετές απαιτήσεις. Αυτές οι απαιτήσεις μπορούν να χαρακτηριστούν είτε ως τεχνικές είτε ως λειτουργικές. Το πιο σημαντικό χαρακτηριστικό μιας λύσης χρονοπρογραμματισμού είναι η ακρίβεια εκτέλεσης των διεργασιών που της ανατίθενται.

Μια μηχανή που παρέχει υψηλή ακρίβεια εγγυάται ότι οι χρονοπραγραμματιζόμενες διεργασίες θα εκτελεστούν στην προκαθορισμένη χρονική στιγμή ή τουλάχιστον με την μικρότερη δυνατή καθυστέρηση, η οποία θα περιλαμβάνεται μέσα σε κάποια όρια ανεκτικότητας. Οι τελικοί χρήστες ενδιαφέρονται κυρίως για τον χρόνο απόκρισης του συστήματος, δηλαδή για το χρόνο που μεσολαβεί από την χρονική στιγμή έναρξης της εκτέλεσης της εργασίας μέχρι την ολοκλήρωση της, καθώς τότε και μόνο τότε θα έχουν τα αποτελέσματα για τα οποία ενδιαφέρονται. Ωστόσο, πιο κρίσιμη θεωρείται η χρονική περίοδος που μεσολαβεί από την έναρξη μιας προγραμματισμένης εργασίας μέχρι τη

στιγμή που πραγματικά ξεκινά η εκτέλεση της, η οποία κατά κανόνα θα πρέπει να είναι της τάξης των μερικών milliseconds.

Στη μελέτη μας, η οποία σχετίζεται με τη δεύτερη κατηγορία, πραγματοποιούμε μια συγκριτική μελέτη των επιδόσεων δυο enterprise χρονοπρογραμματιστών, προσφέροντας έτσι μια παρουσίαση των διαφόρων δυνατοτήτων τους. Οι χρονοπρογραμματιστές στους οποίους θα αναφερθούμε είναι:

- Polos Enterprise Scheduler
- Quartz Enterprise Scheduler

Προτού όμως αναφερθούμε σ'αυτά, θα κάνουμε μία ιστορική αναδρομή στους πρώτους χρονοπρογραμματιστές εργασιών που προσέφεραν λύσεις σε μεγάλη κλίμακα, καθώς και μια συνοπτική αναφορά σε κάποιες πρόσφατες λύσεις χρονοπρογραμματισμού λογισμικού, ερευνητικές και εμπορικές. Τα επόμενα δυο κεφάλαια περιλαμβάνουν μια αναλυτική περιγραφή της αρχιτεκτονικής των δυο υπό μελέτη συστημάτων χρονοπρογραμματισμού, και στη συνέχεια ακολουθεί η περιγραφή των κριτηρίων και των μετρικών αξιολόγησης και η παρουσίαση των συνθηκών διεξαγωγής των σεναρίων.

Το αποτέλεσμα της παρούσας μελέτης αποτυπώνεται στο πέμπτο κεφάλαιο όπου παρουσιάζονται τα αποτελέσματα των μετρήσεων και συγκριτικά διαγράμματα των μετρικών.

1.2 Ιστορική Αναδρομή

1.2.1 Cron – Ο πρώτος job scheduler [33,34]

Στα συστήματα UNIX υπάρχουν εργασίες (tasks, jobs), όπως η διαγραφή προσωρινών αρχείων, που καλό θα ήταν να εκτελούνται αυτόματα ανά τακτά χρονικά διαστήματα χωρίς ο χρήστης να χρειάζεται να παρέμβει. Για παράδειγμα, τα log αρχεία διαφόρων προγραμμάτων αν δεν διαγράφονταν ανά τακτά χρονικά διαστήματα θα δημιουργούσαν πρόβλημα λόγω του ότι θα καταλάμβαναν συνεχώς αυξανόμενο αποθηκευτικό χώρο. Είναι επίσης χρήσιμο τέτοιου είδους εργασίες να εκτελούνται ώρες που το σύστημα βρίσκεται σε όσο το δυνατόν μεγαλύτερη αδράνεια ώστε να μην επηρεάζει την απόδοση του όταν οι χρήστες έχουν ανάγκη από άμεση απόκριση. Το ιδανικό θα ήταν

να μπορέσουν αυτές οι εργασίες να προγραμματιστούν όσο το δυνατόν περισσότερο σε ώρες μη αιχμής, προς αποφυγή της επιπλέον επιβάρυνσης.

Με στόχο όλα τα παραπάνω ο Paul Vixie δημιούργησε το 1987 το χρονοπρογραμματιστή Cron. Χρησιμοποιείται μέσω της εντολής crontab με σκοπό την εκτέλεση εργασιών την ίδια στιγμή κάθε ημέρας, μήνα, ή χρόνου. Κάθε λεπτό το σύστημα ελέγχει ένα αρχείο που ονομάζεται crontab για το αν υπάρχει εργασία που χρειάζεται να εκτελεστεί. Κάθε γραμμή του αρχείου αυτού αποτελείται από έξι πεδία χωρισμένα με κενά. Κάθε πεδίο μπορεί να έχει περισσότερες από μία τιμές χωρισμένες με κόμμα. Με αστερίσκο το πεδίο παίρνει όλες τις δυνατές τιμές. Τα πεδία είναι τα εξής:

1. Λεπτά: τιμές από 0 μέχρι 59
2. Ώρα: τιμές από 0 μέχρι 23
3. Ημέρα του μήνα: τιμές από 1 μέχρι 31
4. Μήνας: τιμές από 1 μέχρι 12
5. Μέρα της εβδομάδας: τιμές από 1 μέχρι 7
6. Εντολή προς εκτέλεση

Παρόλο που ο Cron εκ πρώτης όψεως φαίνεται ένα πολύ χρήσιμο εργαλείο έχει και τα μειονεκτήματά του, τα οποία υπό ορισμένες συνθήκες το καθιστούν ιδιαίτερα αναξιόπιστο. Πρώτο βασικό μειονέκτημα είναι ότι η μέγιστη ακρίβεια με βάση την οποία τρέχει τις προγραμματισμένες εργασίες είναι ένα λεπτό, καθώς κάθε ένα λεπτό ελέγχει το σύστημα το crontab αρχείο πράγμα που πολλές φορές δεν είναι αρκετό. Επίσης κατά την εκτέλεση των εργασιών ο Cron δεσμεύει αρκετούς πόρους του συστήματος για μεγάλο σχετικά χρονικό διάστημα με αποτέλεσμα την μη εύρυθμη λειτουργία του. Το γεγονός ότι δεν διαθέτει δυνατότητα χρησιμοποίησης κατανεμημένων αρχιτεκτονικών είναι ένα σημαντικό μειονέκτημα, καθώς ο Cron θεωρεί ότι ο υπολογιστής στον οποίο τρέχει δεν σταματά ποτέ να λειτουργεί. Αν για κάποιο λόγο ο υπολογιστής κλείσει και κατά τη διάρκεια που είναι κλειστός έχουν προγραμματιστεί να εκτελεστούν κάποιες εργασίες, αυτές δεν πρόκειται να εκτελεστούν ούτε όταν ο υπολογιστής επανέλθει σε κανονική λειτουργία.

Ο Cron Scheduler είναι από τις πιο διαδεδομένες μηχανές χρονοπρογραμματισμού πάνω στην οποία βασιστήκαν και πολλές μεταγενέστερες. Για πολλές όμως εφαρμογές δεν είναι καθόλου επαρκής λόγω των αδυναμιών που αναφέρθηκαν παραπάνω.

1.2.2 Anacron [24]

Ένας χρονοπρογραμματιστής που βασίστηκε πάνω στον Cron είναι ο Anacron, δημιουργός του οποίου είναι ο Sean Perry. Η βελτίωση σε σχέση με τον Cron έγκειται στο ότι δεν θεωρεί ότι το σύστημα είναι συνεχώς ανοιχτό με αποτέλεσμα να μην χάνονται εύκολα προγραμματισμένες εργασίες. Και αυτό όμως δεν παύει να έχει αρκετά προβλήματα στον τρόπο αντιμετώπισης των χρονοπρογραμματισμένων εργασιών. Για παράδειγμα το Anacron μπορεί να τρέχει εργασίες το συχνότερο μία φορά την ημέρα πράγμα που δημιουργεί σοβαρό πρόβλημα σε αρκετές εφαρμογές. Επίσης παρόλο που εγγυάται ότι όλες οι εργασίες που έχουν προγραμματιστεί θα εκτελεστούν, μπορεί να προκύψουν μεγάλες καθυστερήσεις αλλά και πολλές συνεχόμενες εκτελέσεις γεγονός που και αυτό μπορεί να επιβαρύνει πολλές εφαρμογές αλλά και το ίδιο το σύστημα.

1.2.3 Fcron [34]

Μία εξέλιξη του Anacron και κατ' επέκταση του Cron είναι και ο Fcron. Δημιουργός του είναι ο Thibault Godouet. Με πρόσθεση της δυνατότητας να εκτελείται μία εργασία μία φορά ανά συγκεκριμένο χρονικό διάστημα αντιμετωπίζεται το πρόβλημα της διαδοχικής εκτέλεσης της ίδιας εργασίας όταν έχει υπάρξει κάποιο πρόβλημα και το σύστημα για κάποιο χρονικό διάστημα δεν λειτουργούσε. Μία επιπλέον ενδιαφέρουσα δυνατότητα είναι το ότι ο χρήστης μπορεί να καθορίζει το πότε θα τρέξει μία προγραμματισμένη εργασία με βάση το φόρτο του συστήματος εκείνη τη στιγμή θέτοντας κάθε φορά ένα όριο.

Το σημαντικότερο μειονέκτημα όλων των παραπάνω χρονοπρογραμματιστών έγκειται στο γεγονός ότι η εκτέλεση των εργασιών δεν διασφαλίζεται σε περίπτωση δυσλειτουργίας τους. Ένα ακόμα πρόβλημα είναι το ότι μόνο ο διαχειριστής του συστήματος μπορεί να ασχοληθεί με τον χρονοπρογραμματισμό όλων των εργασιών. Τέλος όλοι αφορούν μόνο Unix / Linux πλατφόρμες πράγμα που επηρεάζει την μεταφερσιμότητά τους, σημαντικό στοιχείο στην ποιότητα κάθε σύγχρονου λογισμικού. Ακολουθεί αναφορά σε πιο αναπτυγμένα συστήματα χρονοπρογραμματισμού με δυνατότητες που ανταποκρίνονται περισσότερο στις σύγχρονες απαιτήσεις.

1.3 Σχετικά Συστήματα

1.3.1 JcronTab [26]

Πρόκειται για ένα χρονοπρογραμματιστή που αποτελεί μεταγενέστερη έκδοση του cron. Είναι υλοποιημένος σε Java, χρησιμοποιώντας κυρίως την κλάση Timer και υποστηρίζει ασύγχρονες κλήσεις των διαφόρων εργασιών μέσω νημάτων (threads). Μπορούν να εκτελεστούν διάφοροι τύποι εργασιών όπως java κλάσεις που αντιστοιχούν στην προς εκτέλεση εργασία, Enterprise Java Beans (EJBs), αλλά και εκτελέσιμα προγράμματα. Τα crontables, τα οποία αναλαμβάνουν την οποιαδήποτε διαχείριση των διεργασιών έστω να επιτευχθούν τα ανεκτά επίπεδα ακρίβειας και καθυστέρησης, παίρνουν τα απαιτούμενα δεδομένα είτε από αρχεία δεδομένων είτε από οποιαδήποτε πηγή υλοποιεί μια DataSource διεπαφή (interface). Καθώς ο Jcrontab δεν μπορεί να λειτουργήσει σε συστοιχία υπολογιστών (computer clustering) και δεν υποστηρίζει αποθήκευση της κατάστασης των διεργασιών σε μέσα μόνιμης αποθήκευσης (persistent storage) δεν μπορεί να συμπεριληφθεί στην κατηγορία των enterprise χρονοπρογραμματιστών.

1.3.2 Pulsar [35]

Ο Pulsar είναι ένας χρονοπρογραμματιστής που περιλαμβάνει κάποια enterprise στοιχεία. Είναι υλοποιημένος και αυτός σε Java χρησιμοποιώντας κυρίως την κλάση Timer και κάνει ευρεία χρήση του Java 2 Enterprise Edition (J2EE). Η αρχιτεκτονική του είναι βασισμένη σε ένα ελεγχόμενο κατανεμημένο περιβάλλον. Οι τύποι των διεργασιών που μπορούν να εκτελεστούν είναι java κλάσεις αλλά και Stateless Session Enterprise Java Beans (EJBs). Οι διεργασίες αυτές καταχωρούνται στον χρονοπρογραμματιστή μέσω ενός XML αρχείου, κατάλληλα δομημένου ώστε να περιέχει όλες τις απαραίτητες για την εκτέλεσή τους παραμέτρους. Το κυριότερο μειονέκτημα του είναι ότι δεν μπορεί να λειτουργήσει σε μια συστοιχία υπολογιστών. Σε γενικές γραμμές, μπορεί να ειπωθεί ότι δεν ικανοποιεί όλα τα βασικά κριτήρια ενός enterprise χρονοπρογραμματιστή.

1.3.3 Kronova E-Scheduler [4]

Ο Kronova E-Scheduler, αναπτυγμένος από την εταιρεία Indus Consultancy Services μπορεί να αξιοποιηθεί αποκλειστικά στον χώρο των enterprise εφαρμογών, καθώς

λειτουργεί απαραίτητα μέσα σε κάποιο J2EE container. Το μοναδικό στοιχείο που κληρονομεί από τους cron-like είναι ότι ανά τακτά χρονικά διαστήματα, συγκεκριμένα ανά λεπτό, ελέγχει για το ποιες εργασίες έχουν εκκρεμότητες εκτέλεσης. Το μειονέκτημα του είναι ότι δεν επιτυγχάνει μέγιστη ακρίβεια. Ο Kronova παρέχει υψηλό βαθμό μεταφερσιμότητας εφόσον είναι συμβατός με τους περισσότερους J2EE εξυπηρετητές εφαρμογών. Ο Kronova διαθέτει ένα Application Programming Interface (API), μέσω του οποίου παρέχει μια διεπαφή επικοινωνίας για άλλες εξωτερικές εφαρμογές.

Ο Kronova αποτελείται από μια εφαρμογή που τρέχει πάνω σε έναν εξυπηρετητή εφαρμογών (application server), αλλά και από τρεις εξωτερικές διεργασίες, το σύνολο των οποίων αποτελεί τον Job Agent. Παρέχεται η δυνατότητα μέσα στον εξυπηρετητή εφαρμογών να τρέχουν πολλά στιγμιότυπα (instances), σε καθένα από τα οποία ανατίθεται διαφορετικό σύνολο διεργασιών. Η όλη διαδικασία διαχειρίζεται από τον διαχειριστή, ο οποίος χωρίζει τις εργασίες σε λογικές κατηγορίες και αναθέτει κάθε ομάδα σε ένα στιγμιότυπο. Θα προσπαθήσουμε να κάνουμε μια πρώτη προσέγγιση της αρχιτεκτονικής του Kronova E-Scheduler περιγράφοντας αναλυτικότερα τα υποσυστήματα από τα οποία αυτός αποτελείται:

- Instance: Είναι υπεύθυνο για τις πολιτικές ασφάλειας, τα δικαιώματα των χρηστών. Επίσης, είναι το σημείο στο οποίο βρίσκονται οι εργασίες αλλά και ο χρονοπρογραμματισμός τους.
- Rules Engine: Δέχεται εντολές από τον Job Agent ώστε να εξετάζει ανά τακτά χρονικά διαστήματα (ανά λεπτό) αν υπάρχουν εργασίες που αναμένουν πυροδότηση.

Job Agent: Αποτελεί μια «ελαφριά» Java εργασία που εκτελείται στο ίδιο ή σε άλλο μηχάνημα εκτός του J2EE Container. Σημαντικές δυνατότητες του Kronova E-Scheduler αφορούν διεργασίες. Πιο συγκεκριμένα υποστηρίζονται οι παρακάτω τύποι:

- Command Tasks
- E-Mail Tasks

- EJB Tasks
- RMI Task
- Simple Java Tasks

Ο Kronova E-Scheduler ως χρονοπρογραμματιστής διαθέτει τέσσερις διαφορετικούς τρόπους προγραμματισμού εργασιών:

- Scheduled Jobs

  Οι εργασίες αυτές εκτελούνται σύμφωνα με συγκεκριμένο πρόγραμμα που τους έχει ανατεθεί. Για παράδειγμα, μια εργασία μπορεί να εκτελείται μία φορά την ημέρα, την εβδομάδα, ή και μία φορά το μήνα (π.χ την πρώτη εργάσιμη μέρα του μήνα αυτού). Και άλλοι όροι μπορούν να προστεθούν που μπορούν να εξαρτώνται και από την εργασία.

- Manual Jobs

  Οι εργασίες αυτές αντί να εκτελούνται ανά συγκεκριμένα χρονικά διαστήματα, μπορούν να εκτελεστούν μόνο όταν τους έχει δοθεί ρητή εντολή. Για παράδειγμα, μία εργασία μπορεί να χρειαστεί να τρέχει ανάλογα με την κατάσταση κάποιου εξωτερικού συστήματος.

- Dependency Only Jobs

  Οι εργασίες αυτές εκτελούνται μόνο όταν όλες οι συνθήκες από τις οποίες εξαρτώνται εκπληρωθούν. Μπορεί επίσης να προστεθεί και προκαθορισμένος χρόνος με σκοπό να εκτελεστεί σε συγκεκριμένη χρονική στιγμή η εργασία αφού έχουν εκπληρωθεί όλες οι απαιτούμενες συνθήκες.

- File Triggered Jobs

  Αυτού του είδους οι εργασίες εκτελούνται με την αλλαγή της κατάστασης κάποιου αρχείου. Μπορεί να οριστεί να εκτελεστούν κατά τη δημιουργία, αλλαγή ή διαγραφή ενός αρχείου. Επιπλέον η εκτέλεση μπορεί να βασιστεί σε επεκτάσεις των αρχείων. Για παράδειγμα μία εργασία θα μπορούσε να χρονοπρογραμματιστεί να τρέχει κάθε φορά που ένα αρχείο συγκεκριμένου τύπου δημιουργείται ή αλλάζει.

Βασικό μειονέκτημα του Kronova E-Scheduler είναι το πρόβλημα της ακρίβειας που παρουσιάζεται καθώς οι διεργασίες δεν μπορούν να ειδοποιήσουν οι ίδιες για την εκτέλεση τους.

Πέραν τούτου, δεν υπάρχει η δυνατότητα αποθήκευσης και διατήρησης των δεδομένων κατάστασης κάθε εργασίας. Το πιο σημαντικό, όμως, πρόβλημα παραμένει το ότι αν και ο Kronova E-Scheduler λειτουργεί σε κατανεμημένο περιβάλλον δεν παρουσιάζει δυνατότητα ανάκαμψης από σφάλματα.

1.3.4 Flux Scheduler [10]

Πρόκειται για έναν χρονοπρογραμματιστή υλοποιημένο σε java, στον οποίο προσφέρεται η δυνατότητα διαφορετικών τρόπων πυροδότησης, έτσι ο χρήστης μπορεί να χρονοπρογραμματίσει προγραμματίσει time-driven, event-driven και file-driven διεργασίες. Για τα time-driven tasks δίνεται μεγάλη ευελιξία καθώς υποστηρίζονται και απλά χρονοπρογράμματα αλλά και σύνθετα που λαμβάνουν υπόψη το ανθρώπινο ημερολόγιο. Η μηχανή του Flux λειτουργεί με ακρίβεια δευτερολέπτου που είναι πολύ ικανοποιητική, αξιοποιεί τον πολυνηματισμό προσφέροντας ασύγχρονη εκτέλεση εργασιών και μπορεί να συνδεθεί με οποιαδήποτε βάση δεδομένων.

ΚΕΦΑΛΑΙΟ 2

POLOS ENTERPRISE SCHEDULER

2.1 Βασικές Αρχές

Ο Polos Enterprise Scheduler αναπτύχθηκε στα πλαίσια ερευνητικής μελέτης με βασικό στόχο το σχεδιασμό μίας νέας αρχιτεκτονικής πολύ υψηλής απόδοσης, ακρίβειας, διαθεσιμότητας και αξιοπιστίας, βασισμένης στις τεχνικές δυνατότητες της Java 2.0 Enterprise πλατφόρμας και παίρνει το όνομά της από την ομώνυμη πλατφόρμα παροχής Location Based Services [1], κομμάτι της οποίας αποτελεί . Το λογισμικό είναι αναπτυγμένο στη γλώσσα προγραμματισμού Java, κάνει εκτεταμένη χρήση των standards που παρέχονται από το enterprise framework 5.0 της γλώσσας προγραμματισμού Java και ο J2EE Container που χρησιμοποιήθηκε είναι ο JBoss application server.

Μπορεί να λειτουργήσει πάνω σε όλους τους γνωστούς J2EE application servers (WebLogic, WebSphere, JBoss). Έχει σχεδιαστεί για να λειτουργεί σε συστοιχία εξυπηρετητών, με την προσθήκη ή την αφαίρεση ενός κόμβου με εγκατεστημένο έναν J2EE εξυπηρετητή να είναι μια απλή διαδικασία, καθώς κάθε κόμβος αναγνωρίζεται αυτόματα ως μέλος της συστοιχίας και εγκαθιστά δικτυακά τα συστατικά του χρονοπρογραμματιστή.

Βασικό χαρακτηριστικό της αρχιτεκτονικής του, είναι η αποδέσμευση του συστήματος από τη χρήση βάσης δεδομένων, αλλά και γενικά από οποιουδήποτε μηχανισμού μόνιμης αποθήκευσης μέχρι ενός συγκεκριμένου όγκου εργασίας. Το ρόλο της αποθήκευσης των εργασιών, των χρονοπρογραμμάτων και όλων των απαραίτητων δεδομένων αναλαμβάνει ένας κατανεμημένος μηχανισμός κρυφής μνήμης (cache), όπου τα δεδομένα που περιέχει συγχρονίζονται σε όλους τους κόμβους της συστοιχίας ή σε ομάδες κόμβων. Δηλαδή, η αρχιτεκτονική αυτή, μετατρέπει την καθυστέρηση που οφείλεται στις εντολές ανάγνωσης και γραφής στο δίσκο σε κίνηση στο δίκτυο, το οποίο έχει ως αποτέλεσμα μικρότερες καθυστερήσεις. Η επιλογή αυτή έχει τα εξής πλεονεκτήματα:

- εξάλειψη όλων των μοναδικών σημείων αποτυχίας με το να διατηρείται πάντα η αναγκαία πληροφορία σε κάποιον άλλο κόμβο,

- εξάλειψη των καθυστερήσεων που προκαλούνται από την διαδικασία σύνδεσης και επερωτήσεων στη βάση δεδομένων.

Στην περίπτωση του Polos Enterprise Scheduler, ο κατανεμημένος cache μηχανισμός που χρησιμοποιήθηκε είναι η JBossCache [ref]. Η JBossCache μπορεί να ενσωματωθεί σε οποιονδήποτε JMX (Java Management Extensions) συμβατό εξυπηρετητή με τη μορφή ενός MBean. Υπάρχουν τρία σενάρια χρήσης της σε μία συστοιχία υπολογιστών:

- Η τοπική αποθήκευση, όπου τα δεδομένα αποθηκεύονται τοπικά σε κάθε κόμβο της συστοιχίας χωρίς να παρέχεται ενημέρωση των υπολοίπων κόμβων γι'αυτά.
- Η ασύγχρονη replicated αποθήκευση, η οποία αποθηκεύει τοπικά τα δεδομένα που αλλάζουν αλλά και στέλνει ασύγχρονα μηνύματα ώστε να ενημερώσει τους άλλους κόμβους γι'αυτά.
- Η σύγχρονη replicated αποθήκευση, η οποία, σε κάθε ανανέωση των δεδομένων, κλειδώνει τα δεδομένα σε όλους τους κόμβους προκειμένου να τα ενημερώσει σύγχρονα και μόνο όταν όλοι έχουν ενημερωθεί επιτυχώς άρει το κλείδωμα.

Ο Polos Scheduler εφαρμόζει το τρίτο σενάριο, καθώς εγγυάται μη απώλεια δεδομένων. Ωστόσο μειονεκτεί στο ότι είναι πιο απαιτητικό σε πόρους. Στην περίπτωση που χρησιμοποιείται η σύγχρονη replicated αποθήκευση, μία κλήση αποθήκευσης δεδομένων, έχει σαν αποτέλεσμα να διακοπεί η ροή εκτέλεσης στον κόμβο από τον οποίο ξεκινά, μέχρι να διαδοθεί η αλλαγή σε όλους τους κόμβους, ενώ παράλληλα όλες οι άλλες εργασίες που απαιτούν πρόσβαση σε αυτά τα δεδομένα μπλοκάρονται, καθώς τα κλειδώνει έως ότου «επιστρέψει». Το δεύτερο σενάριο, ενδείκνυται για εφαρμογές του χρονοδρομολογητή που επιζητούν πολύ μεγάλη απόδοση αλλά είναι ανεκτικές σε «βρώμικα» ή ελλιπή δεδομένα.

Τέλος, πρέπει να αναφερθούμε στη τεχνολογία JMX (Java Management Extensions), πάνω στην οποία το είναι βασισμένο το scheduling υποσύστημα. Με αυτήν την τεχνολογία, ένα συστατικό μίας εφαρμογής οργανώνεται από ένα ή περισσότερα Managed Beans ή MBeans, τα οποία εγγράφονται σε ένα MBean Server, ο οποίος λειτουργεί σαν διαχειριστικός πράκτορας διαθέτοντας ένα σύνολο υπηρεσιών για το χειρισμό των MBeans. Ένας επιπλέον λόγος για τον οποίο ο PoloS Scheduler βασίστηκε στη JMX τεχνολογία, είναι ότι αυτή παρέχει ένα νέο managed Java Timer και

ένα νέο μοντέλο συμβάντων. Εφόσον, ο Polos Scheduler εκτελείται μέσα σε ένα διαχειριζόμενο περιβάλλον, προκειμένου να λειτουργεί αξιόπιστα και αποδοτικά, δεν θα μπορούσε να χρησιμοποιηθεί ο κλασσικός Timer της Java, καθώς δεν θα πρέπει να χρησιμοποιείται ποτέ σε managed περιβάλλοντα διότι δημιουργεί νήματα έξω από τον έλεγχο του container.

2.2 Αρχιτεκτονική

Ο Polos Enterprise Scheduler στηρίζει την αρχιτεκτονική του σε τέσσερα βασικά υποσυστήματα, τα οποία περιγράφονται αναλυτικά παρακάτω:

- Management Subsystem

  Είναι το υποσύστημα εκείνο που περιλαμβάνει τις μεθόδους που είναι υπεύθυνες για την διαχείριση των εργασιών. Ουσιαστικά είναι αυτό που συνδέει μια scheduling μηχανή με τις εξωτερικές διεπαφές χρηστών. Το management υποσύστημα αποτελείται από δύο session stateless EJBs, το ένα είναι επιφορτισμένο με την εγγραφή, διαγραφή και αλλαγή εργασιών και το άλλο είναι υπεύθυνο για την ενημέρωση της τρέχουσας κατάστασης των εργασιών, δηλαδή διαθέτει μεθόδους που επιστρέφουν την τρέχουσα κατάσταση τους. Αυτά τα session stateless EJBs λειτουργούν σαν το σημείο διασύνδεσης του scheduler με τους Remote Method Invocation (RMI) Clients.

- Execution Subsystem

  Αναλαμβάνει την ασύγχρονη εκτέλεση των διεργασιών της χρονοπρογραμματισμένης εργασίας. Το Execution υποσύστημα αποτελείται από μια δεξαμενή από Message Driven Beans (MDBs), στα οποία αναθέτει τις διάφορες εργασίες. Η μηχανή εκτέλεσης κάθε κόμβου διαβάζει μηνύματα από την ουρά μηνυμάτων και για κάθε ένα από αυτά τα μηνύματα καλεί ένα message-driven bean και του το παραδίδει. Η κατανάλωση του μηνύματος αντιστοιχεί στην εκτέλεση της εργασίας που περιγράφεται από τα δεδομένα του μηνύματος. Για λόγους βελτίωσης της απόδοσης, διατηρείται μία δεξαμενή (pool) έτοιμων MDBs, τα οποία είναι έτοιμα να λάβουν τέτοια μηνύματα. Σε αυτή τη δεξαμενή τίθεται ένα μέγιστο όριο χωρητικότητας. Σε περίπτωση που το όριο ξεπεραστεί, για κάθε επιπλέον μήνυμα δημιουργείται ένα νέο στιγμιότυπο μέχρι να ξεπεραστεί και το συνολικό μέγιστο όριο, οπότε, σε αυτή την περίπτωση, το

κάθε μήνυμα πρέπει να περιμένει να ελευθερωθεί κάποιο στιγμιότυπο ώστε να καταναλωθεί.

- Scheduling Subsystem

  Αποτελεί την καρδιά του scheduler καθώς είναι υπεύθυνο για τον προγραμματισμό, την πυροδότηση και τον έλεγχο των εργασιών. Αποτελείται από τρία διαφορετικά MBeans:

  - Το ProviderMBean, είναι αυτό στο οποίο εγγράφονται οι εργασίες μέσω μιας μεθόδου εγγραφής και καλεί το αντίστοιχο Session EJB ώστε να καταχωρήσει μια εργασία. Είναι επίσης επιφορτισμένο με το ρόλο της ανάκτησης των εργασιών από την cache. Ο ξεχωριστός ρόλος του έγκειται στο ότι υλοποιεί το πρότυπο σχεδιασμού «singleton», με αποτέλεσμα να εξασφαλίζεται ότι το ProviderMBean είναι αρχικοποιημένο σε όλους τους κόμβους της συστοιχίας αλλά να δουλεύει μόνο σε έναν.
  - Το ManagerMBean, είναι αυτό το οποίο ελέγχει την υπηρεσία του Timer και εσωκλείει ως εσωτερικές κλάσεις τις δομές των εργασιών και των Notification Listeners. Είναι υπεύθυνο για την καταχώρηση των χρονοπρογραμμάτων των εργασιών στον Timer και για την διαχείριση σε πρώτο στάδιο των ειδοποιήσεων (notifications) που στέλνει.
  - Το TaskMBean, το οποίο καλείται από τους Notification Listeners και σχηματίζει τα μηνύματα, τα οποία τοποθετούνται στην ουρά.

- Caching Subsystem

  Είναι η μονάδα αποθήκευσης όλων εκείνων των πληροφοριών που σχετίζονται με τις προς εκτέλεση εργασίες και την πορεία εκτέλεσής τους.

Τα τέσσερα υποσυστήματα που περιγράφηκαν παραπάνω, υπάρχουν σε όλους τους κόμβους (nodes) της συστοιχίας, με εξαίρεση, το scheduling υποσύστημα που είναι ενεργοποιημένο μόνο σε έναν, ο οποίος χαρακτηρίζεται σαν master κόμβος.

Η ενεργοποίηση και λειτουργία του scheduling υποσυστήματος είναι δυναμική, καθώς υπάρχει η δυνατότητα να ανατίθεται η λειτουργία τους υποσυστήματος σε πολύ μικρό χρονικό διάστημα σε οποιοδήποτε από τους άλλους κόμβους της συστοιχίας, σε περίπτωση που η λειτουργία του κόμβου που τρέχει το scheduling υποσύστημα σταματήσει αναπάντεχα. Όπως θα περιγράψουμε στη συνέχεια, μέσα από αυτήν την

διαδικασία μπορεί να επιτραπεί η λειτουργία της ανάκαμψης από σφάλματα (fail-over). Τα υπόλοιπα τρία υποσυστήματα λειτουργούν σε όλους τους κόμβους της συστοιχίας με πανομοιότυπη λειτουργικότητα και συμπεριφορά.

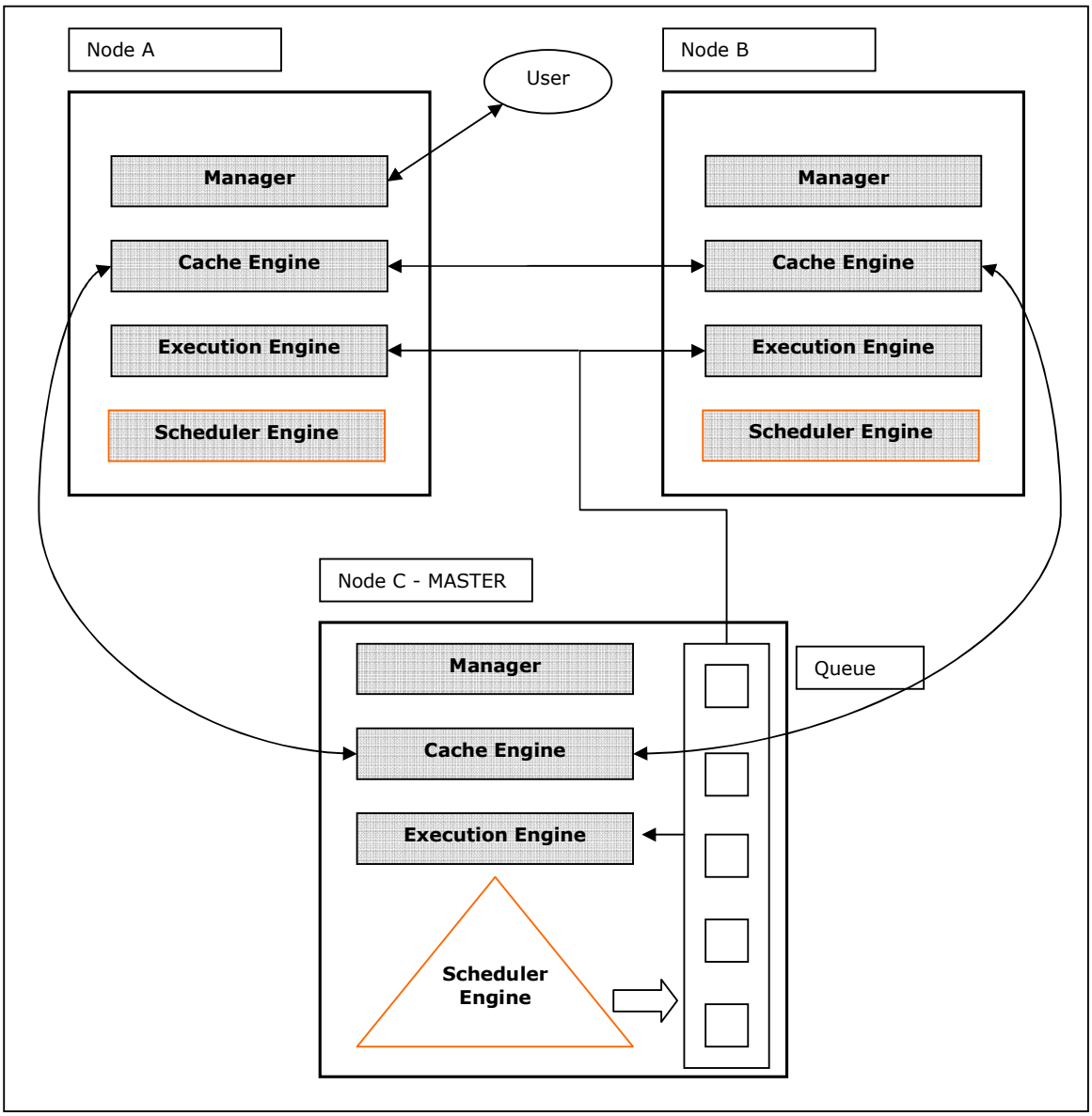
Το Management υποσύστημα, το οποίο λειτουργεί σαν μέσο διασύνδεσης του χρήστη, στην ουσία εξάγει μία διεπαφή για εφαρμογές πελατών, ώστε να υπάρχει η δυνατότητα εισαγωγής, τροποποίησης και τερματισμού μιας διαδικασίας από οποιονδήποτε κόμβο.

Το scheduling υποσύστημα λειτουργεί, όπως αναφέραμε προηγουμένως, σε ένα από τους κόμβους της συστοιχίας. Κύριο μέλημα του είναι η κατανομή της απαιτούμενης επεξεργαστικής ισχύος για την εκτέλεση των εργασιών. Μια ουρά μηνυμάτων λειτουργεί σαν δίοδος επικοινωνίας με τον χρονοδρομολογητή, κάθε μήνυμα που αποθηκεύεται στην ουρά μπορεί να θεωρηθεί σαν μια εντολή για εκτέλεση μιας εργασίας, της οποίας όλα τα δεδομένα της κατάστασης και της λογικής μεταφέρονται πάνω στο μήνυμα αυτό. Ουσιαστικά, η ουρά προωθεί μηνύματα από το scheduling υποσύστημα στο execution υποσύστημα. Αυτό αποτελεί βασικό σημείο διαφοροποίησης από το σύνολο των άλλων enterprise χρονοπρογραμματιστών, καθώς οι υπόλοιποι έχουν τα υπεύθυνα για την εκτέλεση των εργασιών στοιχεία, να χρησιμοποιούν μεθόδους του χρονοπρογραμματιστή ώστε να πάρουν τις προς εκτέλεση εργασίες. Σε αυτό το σημείο, έγκειται η δυνατότητα υλοποίησης της κατανομής του φόρτου (load-balancing), εφόσον παρέχεται η δυνατότητα ύπαρξης τόσων μηχανών εκτέλεσης εργασιών όσοι είναι και οι κόμβοι της συστοιχίας.

Η παραπάνω διάταξη των scheduling και execution υποσυστημάτων έχει σαν πλεονέκτημα ότι είναι δυνατή η συνεχής η λειτουργία τους. Ωστόσο, αυτό που θα εξασφαλίζει το υψηλό επίπεδο αξιοπιστίας είναι η διατήρηση των πληροφοριών που σχετίζονται με τις χρονοπρογραμματιζόμενες διαδικασίες και την εκτέλεση τους, ρόλο τον οποίο έχει αναλάβει η cache μηχανή. Πιο συγκεκριμένα μέσω ενός σύγχρονου κατανεμημένου cache μηχανισμού, και χρησιμοποιώντας τη κύρια μνήμη, διατηρούνται όλα τα δεδομένα που χρειάζονται σε όλους τους κόμβους της συστοιχίας. Έτσι επιτυγχάνεται κάθε κόμβος να είναι αυτόνομος έχοντας πρόσβαση σε οποιαδήποτε πληροφορία αρκεί αυτή να είναι διαθέσιμη και να μην είναι εκείνη τη στιγμή δεσμευμένη από κάποιον άλλο κόμβο. Επιπλέον, δίνεται η δυνατότητα σε έναν κόμβο να αναλάβει ολόκληρο τον χρονοπρογραμματισμό σε περίπτωση που όλοι οι άλλοι κόμβοι έχουν καταρρεύσει. Αυτό οδηγεί σε μία ασφαλή και κατανεμημένη cache μηχανή που δίνει τη δυνατότητα στον χρονοπρογραμματιστή να μην χρειάζεται να παρακολουθεί την

λειτουργία της αφήνοντας την επίβλεψη στον εξυπηρετητή. Για την επικοινωνία μεταξύ των cache υποσυστημάτων απαιτείται η σειριοποίηση και αποσειριοποίηση των αντικειμένων που αφορούν τα δεδομένα, γεγονός που οδηγεί σε δαπανηρές λύσεις.

Στο ακόλουθο σχήμα παρουσιάζεται η αρχιτεκτονική του Polos Enterprise Scheduler σε μια συστοιχία τριών κόμβων.



Σχήμα 1: Αρχιτεκτονική του PoLoS Enterprise Scheduler.

Το scheduling υποσύστημα τρέχει στον κόμβο C, επομένως από αυτό τον κόμβο πηγάζουν οι πυροδοτήσεις των χρονοπρογραμματιζόμενων εργασιών. Αυτό επιτυγχάνεται με την γράψιμο μηνυμάτων στην ουρά. Ο κόμβος αυτός αποτελεί τον

Master κόμβο της συστοιχίας. Τα execution υποσυστήματα, με τη σειρά τους,

διαβάζουν τα διάφορα μηνύματα και δημιουργούν για κάθε μήνυμα ένα νήμα για την ασύγχρονη εκτέλεση της εργασίας που περιγράφεται από το μήνυμα που έχουν παραλάβει από την ουρά.

Ο κάθε χρήστης, επικοινωνεί μέσω εξωτερικών διεπαφών με έναν κόμβο της συστοιχίας. Ο αντίστοιχος manager του κόμβου ενημερώνει τον χρονοδρομολογητή, ο οποίος τρέχει στον Master κόμβο C για όλες τις ενέργειες του χρήστη. Η cache ενημερώνεται όχι μόνο για κάθε εισαγωγή, τροποποίηση, διαγραφή μιας εργασίας από την ουρά αλλά και για το αποτέλεσμα της εκτέλεσης της.

Τέλος, το σύστημα Polos Enterprise λειτουργεί αυτόνομα μέσα σε ένα J2EE Container ενός εξυπηρετητή,. Μοναδική απαίτηση, η οποία υπάρχει σε κάθε κατανεμημένο περιβάλλον, είναι ο συγχρονισμός των ρολογιών των κόμβων της συστοιχίας όταν οι κόμβοι τρέχουν σε ξεχωριστά μηχανήματα. Αφού αναφερθήκαμε σε βασικά σημεία της αρχιτεκτονικής του, στη συνέχεια θα προχωρήσουμε σε μία πιο λεπτομερή ανάλυση της λειτουργίας και αρχιτεκτονικής του.

2.3 Ανάλυση

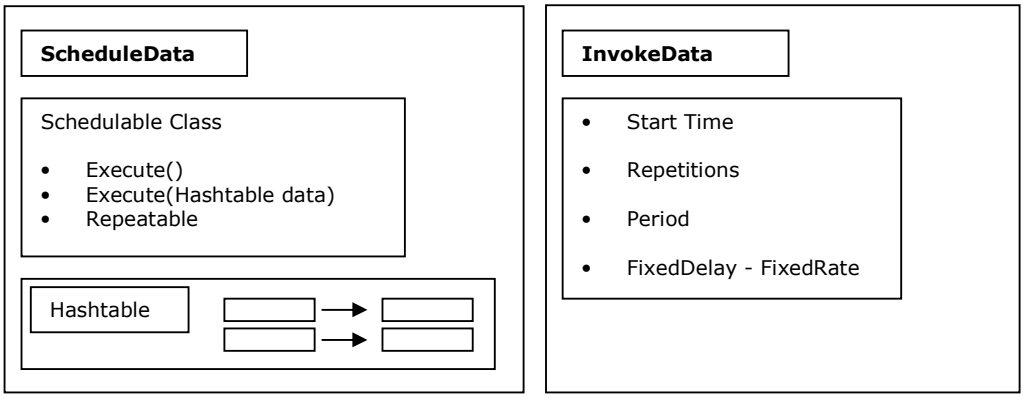
2.3.1 Η βασική λειτουργία του Scheduling υποσυστήματος

Η κάθε εργασία που δίνεται στον Polos Enterprise Scheduler περιγράφεται από ένα αντικείμενο τύπου ScheduleData, το οποίο είναι ένα JavaBean αντικείμενο με δυο μεταβλητές – μέλη, το στιγμιότυπο της Java κλάσης, η οποία υλοποιεί τη Schedulable διεπαφή και αποτελεί την χρονοπρογραμματιζόμενη εργασία και ένας χάρτης κλειδιών σε τιμές που αναπαρίσταται από τη Hashtable Java κλάση. Η Schedulable διεπαφή ορίζει τις μεθόδους execute() και execute(hashtable) και πρέπει να υλοποιείται από κάθε κλάση που περιγράφει μια εργασία. Το Hashtable είναι ένας κατακερματισμένος πίνακας που περιέχει τα δεδομένα που χρειάζεται η εργασία και δίνει τη δυνατότητα σε αυτή να τα τροποποιεί και να τα επαναχρησιμοποιεί μεταξύ των διαδοχικών της εκτελέσεων.

Η μορφή των χρονοπρογραμμάτων που προσφέρονται είναι η απλούστερη δυνατή καθώς πρόκειται για μια λύση που βρίσκεται ακόμα σε ερευνητικά πλαίσια, και για το σκοπό αυτό περαιτέρω λειτουργικότητα όσον αφορά τις δυνατότητες

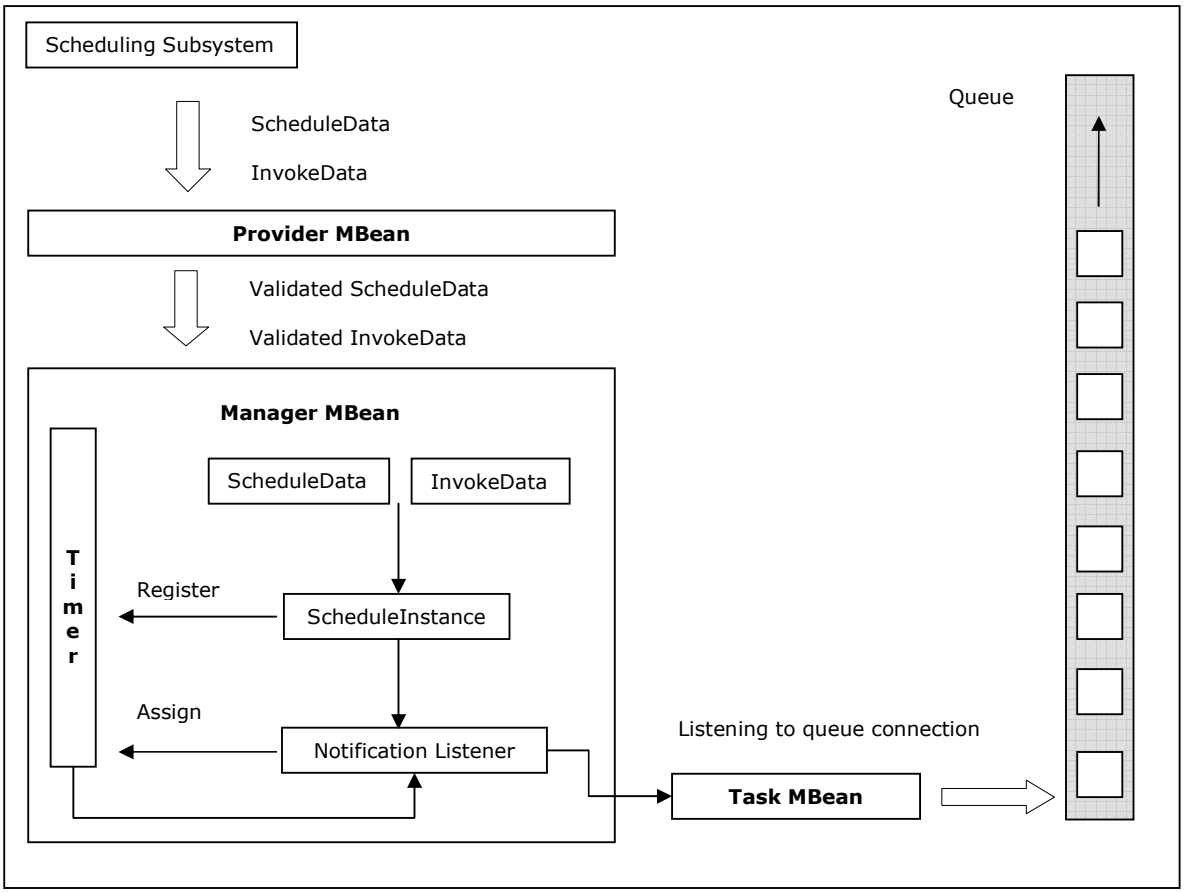
χρονοπρογραμματισμού είναι περιπτή. Το κάθε χρονοπρόγραμμα αναπαρίσταται από ένα Java Bean με παραμέτρους:

- τον χρόνο έναρξης,
- το μήκος της περιόδου,
- το πλήθος των επαναλήψεων,
- το τύπο της χρονικής συμπεριφοράς, η οποία μπορεί να είναι:
  - fixedDelay, όπου σε περίπτωση καθυστέρησης στην εκτέλεση μιας επανάληψης, η καθυστέρηση αυτή μεταδίδεται και στις μελλοντικές εκτελέσεις, αρχίζοντας την εκτέλεση της αμέσως επόμενης επανάληψης κανονικά μετά την πάροδο της περιόδου.
  - fixedRate, όπου σε περίπτωση καθυστέρησης, οι επόμενες επαναλήψεις εκτελούνται πολύ γρήγορα μέχρι η ακολουθία των εκτελέσεων να συμβαδίσει ξανά με το αρχικό χρονοπρόγραμμα



Σχήμα 2: Τα Java Beans που περιγράφουν Jobs & Triggers.

Στο σχήμα 3 παρουσιάζεται η αρχιτεκτονική του scheduling υποσυστήματος και αναπαρίσταται μια λειτουργία της εγγραφής και εκτέλεσης μίας εργασίας, η οποία περιγράφεται αναλυτικά παρακάτω.



Σχήμα 3: Το scheduling υποσύστημα του Polos Scheduler – Τυπική διαδικασία εγγραφής μίας εργασίας.

Ο Manager είναι αυτός που ρυθμίζει τα πάντα, καθώς σε αυτόν εγγράφονται ένας ή περισσότεροι Providers, οι οποίοι τον τροφοδοτούν με εργασίες και χρονοπρογράμματα. Προς το παρόν, ορίζεται μόνο ένας Provider ο οποίος είναι ένα MBean και διαθέτει τις μεθόδους start() και stop(), μέσω των οποίων καθορίζεται η έναρξη και η λήξη της λειτουργίας του. Επιπρόσθετες μέθοδοι του Provider είναι η insertSchedule (InvokeData, ScheduleData), η οποία καλείται από ένα Session EJB, ώστε να προστεθούν τα δεδομένα μιας εργασίας και του χρονοπρογράμματος της αλλά και η loadFromCache(), η οποία καλείται αυτόματα κάθε φορά που ο Provider ξεκινάει και καλεί με τη σειρά της την insertSchedule για κάθε εργασία που υπάρχει στην cache.

Η insertSchedule ελέγχει αν ο Manager είναι διαθέσιμος, και μεταφέρεται στον Manager μέσω της addSchedule (InvokeData, ScheduleData, Boolean) μεθόδου. Η παράμετρος

τύπου Boolean καθορίζει το εάν τα δεδομένα αυτά προήλθαν απευθείας από το χρήστη ή ήδη βρίσκονταν στην cache.

Στη συνέχεια, ο Manager παίρνει τα InvokeData και ScheduleData και τα τοποθετεί σε μία νέα δομή που ονομάζεται ScheduleInstance. Αυτή, όπως υποδηλώνει το όνομά της, περιλαμβάνει όλα τα δεδομένα για κάθε εργασία, δηλαδή δεν περιέχει μόνο τις αρχικές πληροφορίες χρονοπρογραμματισμού αλλά και στοιχεία σχετικά με την τρέχουσα κατάσταση της εργασίας, όπως το σε ποια επανάληψη βρίσκεται. Όταν δημιουργείται ένα ScheduleInstance του ανατίθεται ένας auto-generated μοναδικός αριθμός που αποτελεί την ταυτότητα, με την οποία αποθηκεύεται στην cache. Στη συνέχεια, ακολουθεί η εγγραφή στον JMX Timer ενώ ταυτόχρονα δημιουργείται και ένας Notification Listener ο οποίος ακούει μόνο τα Notifications που αντιστοιχούν στο συγκεκριμένο ScheduleInstance, μέσω ενός κατάλληλου φίλτρου.

Ο Notification Listener βρίσκεται σε κατάσταση αναμονής έως ότου ειδοποιηθεί ώστε να πυροδοτήσει μία εργασία. Όταν αυτός ειδοποιείται, αρχικά ελέγχει αν όλα τα MBeans βρίσκονται σε λειτουργία ή κάποιο έχει σταματήσει. Σε περίπτωση που λειτουργούν όλα, ελέγχει αν η εργασία είναι ακόμη έγκυρη ή έχει αλλάξει από το χρήστη. Εφόσον είναι ακόμα έγκυρη, καλεί το TaskMBean, στο οποίο και παραδίδει την προς εκτέλεση εργασία, δηλαδή παραδίδει το ScheduleData και την κλάση που αντιστοιχεί στην εργασία. Αν η κλήση του TaskMBean επιστρέψει επιτυχώς, τότε ενημερώνεται το αντίστοιχο ScheduleInstance, καταγράφει το ότι η εργασία πρόκειται να εκτελεστεί και ακολούθως μειώνει τον αριθμό των εναπομεινάντων επαναλήψεων κατά ένα. Το TaskMBean, όμως, έχει ακόμα ένα ιδιαίτερο ρόλο, ο οποίος έγκειται στο ότι διατηρεί μία μόνιμη σύνδεση με την ουρά μηνυμάτων, ανιχνεύει τυχόν αποτυχία της και σε μια τέτοια περίπτωση προσπαθεί να επανασυνδεθεί. Τα TaskMBeans είναι μόνιμα συνδεδεμένα με την master ουρά και ανιχνεύουν την καταστροφή της υλοποιώντας μια ExceptionListener διεπαφή. Μετά από μία τέτοια ανίχνευση, προσπαθούν να εντοπίσουν το νέο master κόμβο και να συνδεθούν στη νέα ουρά. Η διαδικασία επανασύνδεσης είναι σύγχρονη, δηλαδή εκτελείται μόνη της και ολοκληρωμένα κάθε φορά για το ίδιο αντικείμενο, στην περίπτωσή μας το MBean, με αποτέλεσμα να παρουσιάζεται πολύ μικρότερο κόστος επανασύνδεσης αφού ακόμα και υπάρχουν, για παράδειγμα, 1000 κλήσεις στο TaskMBean που αποτυγχάνουν, μία μόνο επαναλαμβανόμενη προσπάθεια επανασύνδεσης λαμβάνει χώρα.

Η ουρά μηνυμάτων, ρόλος της οποίας είναι να υποδέχεται τις προς εκτέλεση εργασίες και να τις μοιράζει στα κατανεμημένα MDBs, είναι μέρος του scheduling υποσυστήματος και επομένως υπακούει στο πρότυπο «singleton», δηλαδή υπάρχει μόνο στον master κόμβο. Σε περίπτωση αποτυχίας αυτού, μια νέα ουρά δημιουργείται στον κόμβο που πια λειτουργεί σαν master.

Το Scheduling υποσύστημα, όπως έχουμε ήδη δει, είναι ουσιαστικά μία δεξαμενή από MDBs, ρόλος των οποίων είναι να παίρνουν τα διάφορα μηνύματα που βρίσκονται στην ουρά και να εκτελούν τις αντίστοιχες εργασίες. Μέσω αυτής της διαδικασίας, πραγματοποιείται η κατανομή του φόρτου εργασίας που οφείλεται στην εκτέλεση των εργασιών. Η κατανομή αυτή εξαρτάται από τα εξής τρία χαρακτηριστικά:

- ο τρόπος πρόσβασης στην ουρά,
- ο τύπος των μηνυμάτων,
- ο αλγόριθμος κατανομής αυτών μέσα στη συστοιχία των εξυπηρετών.

Ενώ το μέγεθος της δεξαμενής των MDBs, ο μέγιστος αριθμός MDB στιγμιότυπων και η διατήρηση της σύνδεσης με την ουρά, καθορίζονται από παραμέτρους που έχει ορίσει ο διαχειριστής και τις διαχειρίζεται ο EJB Container.

Όπως έχει ήδη αναφερθεί, όταν υπάρχει μία συστοιχία Ν κόμβων, η ουρά μηνυμάτων θα οριστεί μόνο σε έναν από τους κόμβους της συστοιχίας, στον Master κόμβο. Επομένως, πρέπει οι EJB Containers όλων των υπολοίπων κόμβων να έχουν την δυνατότητα να εντοπίσουν την ουρά στον Master κόμβο αλλά και να συνδεθούν μαζί της. Το ποιος θα είναι ο Master κόμβος είναι μια δυναμική επιλογή που δεν είναι προκαθορισμένη κατά την εγκατάσταση, αλλά, και σε περίπτωση που ο Master κόμβος σταματήσει να λειτουργεί, το ποιος θα αναλάβει αυτό το ρόλο είναι, ομοίως, μια τυχαία επιλογή.

Ο Polos Scheduler βασίζεται σε ένα μηχανισμό αυτόματης ανακάλυψης των διαφόρων πόρων που χρησιμοποιεί χωρίς να απαιτείται παρέμβαση από τους χρήστες για τον εντοπισμό τους. Επομένως, η ανίχνευση της ουράς μηνυμάτων πραγματοποιείται κάνοντας χρήση του high availability ή global JNDI ευρετηρίου. Πρόκειται για ένα ευρετήριο προσπελάσιμο από όλους τους κόμβους της συστοιχίας στο οποίο καταχωρούνται αντικείμενα κάτω από ένα συγκεκριμένο όνομα. Το ευρετήριο αυτό υφίσταται όσο υπάρχει ένας τουλάχιστον κόμβος της συστοιχίας σε λειτουργία. Η λογική

αναζήτησης σε αυτό το ευρετήριο είναι η ακόλουθη :

1. Αν το ζητούμενο όνομα υπάρχει στο cluster-wide δέντρο τότε επιστρέφεται.
2. Αν το ζητούμενο όνομα δεν υπάρχει στο cluster-wide δέντρο, τότε η αναζήτηση ανατίθεται στο τοπικό Java Naming Directory InterfaceJNDI του κόμβου από τον οποίο έχει αρχίσει και αν βρεθεί εκεί επιστρέφεται.
3. Εάν δεν υπάρχει ούτε στο τοπικό JNDI του κόμβου από τον οποίο έχει αρχίσει, η global JNDI υπηρεσία ρωτάει κάθε έναν από τους κόμβους της συστοιχίας εάν το τοπικό τους ευρετήριο διαθέτει ένα τέτοιο όνομα και επιστρέφει τη πρώτη επιτυχή αναζήτηση, εάν υπάρχει τέτοια.
4. Εάν κανένα τοπικό ευρετήριο δεν διαθέτει το ζητούμενο όνομα, τότε προκαλείται ένα NameNotFoundException.

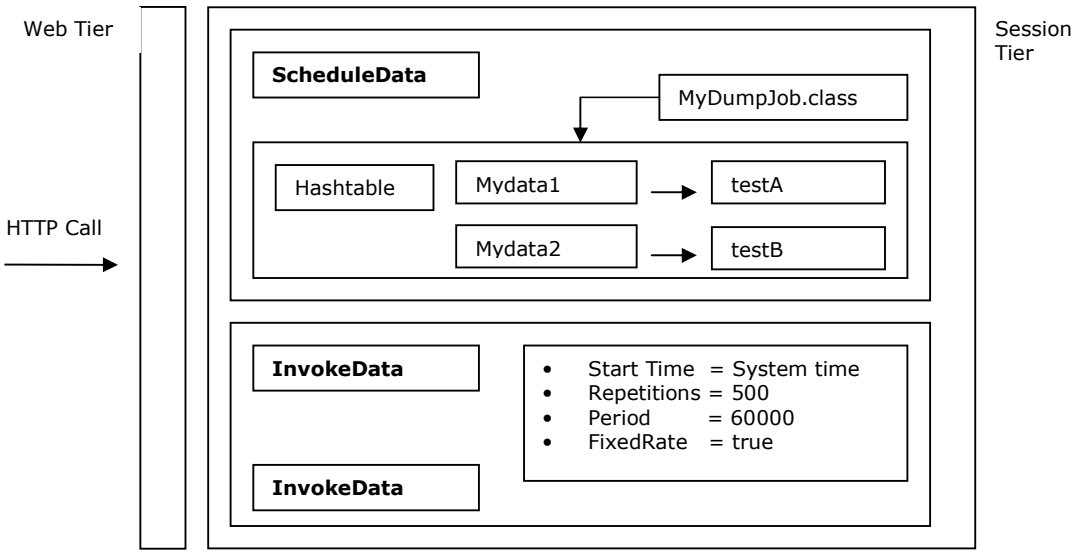
Επομένως, η ουρά μηνυμάτων αντιστοιχεί σε ένα μοναδικό όνομα στο τοπικό ευρετήριο του Master κόμβου. Σύμφωνα με τον παραπάνω αλγόριθμο αναζήτησης, ο EJB Container του Master κόμβου εντοπίζει την ουρά στο 2^{ο} βήμα του αλγορίθμου και οι EJB Container των N-1 υπολοίπων κόμβων στο 3^{ο} βήμα του. Όταν ο Master κόμβος σταματήσει να λειτουργεί, οι EJB Container των N-1 υπολοίπων κόμβων ξεκινούν μια περιοδική προσπάθεια νέου εντοπισμού, και εφόσον εντοπιστεί, προσπαθούν να επανασυνδεθούν με αυτή. Η όλη διαδικασία διαρκεί λίγα δευτερόλεπτα.

Ο τύπος αυτών των μηνυμάτων είναι ο ObjectMessage που ορίζεται στο JMS API και κάθε μήνυμα εσωκλείει ένα serializable αντικείμενο. Γι'αυτό το λόγο υπάρχει η απαίτηση το ScheduleData, δηλαδή το αντικείμενο που μεταφέρεται στο κάθε μήνυμα να είναι serializable, απαίτηση η οποία ισχύει και για την Schedulable διεπαφή αλλά και για τους τύπους δεδομένων που περιέχονται στο Hashtable.

2.3.2 Web-Tier & Session Facade

Το Management υποσύστημα αποτελείται από δύο Stateless Session EJBs, τα JobRegisterBean και JobMonitorBean. Η λειτουργικότητα τους σχετίζεται με την προσθήκη, αφαίρεση και οποιοδήποτε τροποποίηση εργασιών και χρονοπρογραμμάτων αλλά και την ενημέρωση της τρέχουσας κατάστασης των εργασιών, αξιοποιώντας τα tracking data.

Ο client ο οποίος χρησιμοποιείται στο Polos framework είναι ένας web client, που παρέχει web σελίδες, τροφοδοτώντας τον χρονοπρογραμματιστή με τις κατάλληλες παραμέτρους μέσω GET και POST HTTP κλήσεων. Μια τέτοια κλήση οδηγεί στην εγγραφή μίας νέας εργασίας μαζί με το χρονοπρόγραμμα της. Η κλήση αυτή περιλαμβάνει κυρίως χρονικές παραμέτρους εκτέλεσης, το όνομα της κλάσης που την υλοποιεί, αλλά και κάποιες επιπλέον παραμέτρους οι οποίες θεωρούνται ως δεδομένα της εργασίας.



Σχήμα 4: Το αποτέλεσμα της παραπάνω HTTP κλήσης , στο Session επίπεδο

Η κλάση MyDumpJob, η οποία είναι η χρονοπρογραμματιζόμενη εργασία, πρέπει να είναι διαθέσιμη στον ClassLoader της JVM του scheduler. Αυτό μπορεί να γίνει τοποθετώντας το σε κάποιο φάκελο του εξυπηρετητή την αντίστοιχη κλάση ή παρέχοντας το αρχείο της.

Σε κάθε κόμβο της συστοιχίας είναι διαθέσιμο το web tier το οποίο επικοινωνεί με το Session Tier μέσω των τοπικών διεπαφών των EJBs. Επομένως, μέσω του web-interface, ένας χρήστης μπορεί να συνδεθεί με τον browser του ή κάποιο άλλο εργαλείο σε οποιονδήποτε κόμβο της συστοιχίας.

2.3.3 Load Balancing

Η κατανομή του φόρτου (load balancing) στον Polos Enterprise Scheduler βασίζεται στην κατανεμημένη λειτουργία του scheduling υποσυστήματος. Υπάρχουν δύο αλγόριθμοι που χρησιμοποιούνται για την κατανομή του φόρτου εργασίας:

- High Load, με βάση τον αλγόριθμο αυτό τα μηνύματα αρχικά παραδίδονται στο execution υποσύστημα του κόμβου στον οποίο βρίσκεται η ουρά, και μόνο όταν η δεξαμενή γεμίσει, οι άλλοι κόμβοι ξεκινούν να καταναλώνουν μηνύματα από την ουρά.
- Competing Servers, ο αλγόριθμος αυτός στηρίζεται στην round-robin λογική, με βάση την οποία τα μηνύματα ισοκατανέμονται μεταξύ των κόμβων της συστοιχίας. Επομένως, με τη λογική αυτή, όλοι οι κόμβοι της συστοιχίας θεωρούνται ισάξιοι ανταγωνιστές στην κατανάλωση μηνυμάτων.

2.3.4 Fault-Tolerance

Η δυνατότητα του scheduling υποσυστήματος να ενεργοποιείται αυτόματα σε οποιονδήποτε κόμβο μιας συστοιχίας εξασφαλίζει την ανάκαμψη από μια αποτυχία (fail-over) και την ανοχή στα σφάλματα εκτέλεσης (fault- tolerance). Αυτό ισχύει στην περίπτωση που ο Master κόμβος τίθεται εκτός λειτουργίας, διαφορετικά σε κάθε άλλη περίπτωση κατά την οποία υπάρχει σφάλμα σε ένα περιφερειακό κόμβο, τότε αυτός απλά σταματάει να καταναλώνει μηνύματα και επομένως να εκτελεί εργασίες. Οι εργασίες που προορίζονταν για αυτό τον κόμβο, θα εκτελεστούν από τους υπόλοιπους κόμβους της συστοιχίας.

Το γεγονός ότι οι εργασίες και τα χρονοπρογράμματά τους φυλάσσονται στην cache εγγυάται ότι δεν θα χαθεί κάποια προγραμματιζόμενη εκτέλεση ως επακόλουθο του τερματισμού ενός κόμβου.

Όλος αυτός ο σχεδιασμός εξασφαλίζει τη διαρκή διαθεσιμότητα του χρονοπρογραμματιστή προς τους χρήστες αλλά και την διασφάλιση των εργασιών και των χρονοπρογραμμάτων ώστε να μην επηρεάζονται από τυχόν σφάλματα σε κόμβους, είτε είναι ο Master είτε όχι.

Το σύστημα αυτό φροντίζει ακόμα και για τις εργασίες που ήδη εκτελούνται στον κόμβο που αποτυγχάνει αλλά και για εκείνες που βρίσκονται στην ουρά μηνυμάτων στην περίπτωση που ο κόμβος που αποτυγχάνει είναι ο Master.

Όσον αφορά τις εργασίες της πρώτης κατηγορίας η διαδικασία που ακολουθείται είναι αρκετά σύνθετη. Τα MDBs, προαιρετικά, μπορούν να ενημερώνουν τα ScheduleInstance στην cache σχετικά με το αν εκτελέστηκε επιτυχώς η εργασία ή όχι. Ωστόσο, ακόμα και αν παρέχεται ενημέρωση για το αν εκτελέστηκε η εργασία ή όχι, δεν υπάρχει πληροφόρηση για άλλα χαρακτηριστικά, όπως πόση ήταν η διάρκεια της εκτέλεσης, με αποτέλεσμα να μην γνωρίζουμε πότε πρέπει να λάβουμε την σχετική ενημέρωση. Αλλά και στην περίπτωση όπου εξαρχής έχει καθοριστεί η διάρκεια ή ακόμα και αν υπάρχει ένα ανώτατο χρονικό πλαίσιο μέσα στο οποίο πρέπει να έχει ολοκληρωθεί, αγνοούμε στοιχεία όπως μέχρι ποιο σημείο είχε φτάσει η εκτέλεση όταν έγινε ο τερματισμός. Σε αυτή την περίπτωση υιοθετείται μια απλοϊκή λύση, σύμφωνα με την οποία ορίζεται ένα χρονικό περιθώριο ίσο με την περίοδο της εργασίας μέσα στο οποίο πρέπει να υπάρχει η ενημέρωση εάν η εργασία έχει εκτελεστεί. Στην περίπτωση κατά την οποία ο τύπος της χρονικής συμπεριφοράς έχει οριστεί σαν fixedDelay και μόνο τότε, η απώλεια επιβεβαίωσης οδηγεί σε επανεκτέλεση.

Όσον αφορά τις εργασίες που βρίσκονται με τη μορφή μηνυμάτων μέσα στην ουρά του Master κόμβου όταν αυτός αποτυγχάνει, ο χρονοδρομολογητής παρέχει μία επιπλέον ενημέρωση της cache. Έτσι, τα MDBs μπορούν επίσης προαιρετικά να ρυθμιστούν ώστε κάθε φορά που αναλαμβάνουν μία εργασία, το αντίστοιχο ScheduleInstance στην cache να ενημερώνεται. Το Provider MBean είναι αυτό που αναλαμβάνειτην επεξεργασία των δεδομένων στην cache, μέσω της οποίας επιτυγχάνεται η ανοχή στα σφάλματα και το οποίο ξεκινάει την διαδικασία ανάκαμψης για δύο λόγους:

- ειδοποιείται από τον εξυπηρετητή, ότι προέκυψε κάποια αλλαγή στη συστοιχία στη περίπτωση απώλειας ενός τυχαίου κόμβου
- ενεργοποιείται αυτόματα στη περίπτωση απώλειας του Master κόμβου.

Το Provider MBean, κατά τη διαδικασία της ανάκαμψης, ανακτά όλα τα ScheduleInstance που είναι στη cache, τα παραδίδει στον Manager με την οδηγία, κάθε ένα από αυτά τα στιγμιότυπα να ξεκινήσει σε recovery mode. Η διαδικασία αυτή οδηγεί σε πολύ καλή αξιοπιστία, καθώς η αποτυχία ενός τυχαίου κόμβου επηρεάζει την

ακρίβεια μόνο μικρού μέρους των εκτελούμενων εργασιών και καμία εργασία που δεν βρίσκονταν υπό εκτέλεση.

ΚΕΦΑΛΑΙΟ 3

QUARTZ SCHEDULER

3.1 Βασικές Αρχές

Όπως διαπιστώσαμε από την αναφορά μας στις διάφορες λύσεις χρονοπρογραμματισμού, εμπορικές και μη, υπάρχουν αρκετοί χρονοπρογραμματιστές υλοποιημένοι σε java, οι οποίοι παρουσιάζουν enterprise χαρακτηριστικά. Στην παρούσα χρονική στιγμή, κυρίαρχος στο χώρο των λύσεων χρονοπρογραμματισμού είναι ο Quartz. Ο Quartz είναι ένα σύστημα χρονοπρογραμματισμού που παρουσιάστηκε την άνοιξη του 2001 από τον James House. Η ευρεία χρήση του επιτεύχθηκε μετά από την ένταξη του στην οικογένεια των Open Symphony λογισμικών. Μέσα σε αυτά τα χρόνια, από το 2001 μέχρι σήμερα, έχουν παρουσιαστεί διάφορες εκδόσεις του με πιο σύγχρονη την 1.6, η οποία χρησιμοποιήθηκε στα πλαίσια αυτής της μελέτης.

Ο Quartz είναι γραμμένος αποκλειστικά σε Java, σχεδιασμένος να μπορεί να αξιοποιηθεί τόσο σε J2SE όσο και σε J2EE εφαρμογές. Προσφέρει μεγάλες δυνατότητες στο σχεδιασμό χρονοδιαγραμμάτων εκτέλεσης διεργασιών, ξεκινώντας από τις πιο απλές, όπως τον προγραμματισμό της εκτέλεσης μίας εργασίας κάθε μέρα σε τακτή ώρα, και καταλήγοντας στις πιο περίπλοκες, όπως μια εργασία η οποία πρέπει να εκτελεστεί την τελευταία Παρασκευή κάθε μήνα από το έτος 2002 μέχρι το 2005. Σε γενικές γραμμές, οι εργασίες είναι προγραμματισμένες να τρέξουν όταν ένα trigger πραγματοποιείται. Τα triggers μπορούν να δημιουργηθούν με σχεδόν οποιονδήποτε συνδυασμό των παρακάτω:

- Σε συγκεκριμένη χρονική στιγμή μέσα στην ημέρα.
- Σε συγκεκριμένες μέρες της εβδομάδας.
- Σε συγκεκριμένες μέρες του μήνα.
- Σε συγκεκριμένες μέρες ενός καταγεγραμμένου ημερολογίου.
- Να επαναλαμβάνεται συγκεκριμένο αριθμό φορών.
- Να επαναλαμβάνεται μέχρι κάποια συγκεκριμένη χρονική στιγμή ή ημερομηνία.
- Να επαναλαμβάνεται αόριστα, ανά περιοδικά διαστήματα.

Δεν υπάρχει κάποιος περιορισμός στον τρόπο εφαρμογής του Quartz καθώς προσφέρονται όλες οι ακόλουθες δυνατότητες:

- Μπορεί να ενσωματωθεί μέσα σε οποιαδήποτε άλλη εφαρμογή.
- Μπορεί να ενσωματωθεί μέσα σε οποιαδήποτε εφαρμογή η οποία τρέχει μέσα σε έναν εξυπηρετητή εφαρμογών και παίρνει μέρος σε συναλλαγές με κάποια σχεσιακή βάση.
- Μπορεί να εκτελεστεί σαν ανεξάρτητη εφαρμογή (χρησιμοποιώντας δικό του JVM) και να προσπελαστεί μέσω του RMI (Remote Method Invocation).
- Μπορεί να τρέχει σαν συστοιχία (cluster) ανεξάρτητων εφαρμογών προσφέροντας ανάκαμψη από σφάλματα και κατανομή φόρτου, έννοιες στις οποίες θα αναφερθούμε παρακάτω.

Περιέχει περίπου 300 Java κλάσεις και διεπαφές και είναι οργανωμένο σε 12 πακέτα. Παρόλο που το μέγεθος του κώδικα δεν μπορεί να αποτελέσει χαρακτηριστικό μέτρησης της ποιότητας του χρονοπρογραμματιστή, ωστόσο, είναι μια σαφής ένδειξη της λειτουργικότητας που προσφέρεται.

Η καρδιά του Quartz framework είναι ο Scheduler. Ο Scheduler είναι υπεύθυνος για την διαχείριση του περιβάλλοντος εκτέλεσης (runtime environment) της Quartz εφαρμογής. Δεν κάνει όλη την εργασία μόνος του αλλά βασίζεται σε σημαντικά άλλα τμήματα του συστήματος. Ο Quartz βασίζεται πολύ στα νήματα και την διαχείρισή τους. Προκειμένου να εγγυηθεί κλιμάκωση (scalability), ο Quartz βασίζεται σε πολυνηματική αρχιτεκτονική (multithreading). Κατά την έναρξη του, το framework αρχικοποιεί μια ομάδα από νήματα εργάτες (worker threads), τα οποία χρησιμοποιούνται από το χρονοπρογραμματιστή για την εκτέλεση των εργασιών. Με αυτό τον τρόπο, επιτυγχάνεται η παράλληλη εκτέλεση πολλών εργασιών.

Μία εργασία στην ορολογία του Quartz, είναι απλά μία Java κλάση που εκτελεί μια δειργασία, το οποίο μπορεί να είναι οτιδήποτε μπορεί να προγραμματιστεί σε java κώδικα. Η μόνη απαίτηση είναι ότι θα πρέπει η κλάση αυτή να υλοποιεί την org.quartz.job διεπαφή, η οποία περιέχει μια και μόνο μέθοδο, την execute(), και να προκαλεί ένα JobExecutionException σε περίπτωση σοβαρού σφάλματος.

Όταν ο Quartz καλεί τη μέθοδο execute(), της περνάει σαν όρισμα το org.quartz.JobExecutionContext, το οποίο περιλαμβάνει πληθώρα πληροφοριών για το

περιβάλλον εκτέλεσης μέσα στο οποίο εκτελείται η εργασία. Το JobExecutionContext προσφέρει πρόσβαση σε στοιχεία σχετικά με το χρονοπρογραμματιστή, την εργασία, το trigger και πολλά ακόμα. Στις περισσότερες περιπτώσεις, το JobExecutionContext χρησιμοποιείται για να υπάρχει πρόσβαση στην org.quartz.JobDetails κλάση. Η JobDetails κρατά λεπτομερή στοιχεία για την εργασία, συμπεριλαμβανομένης της ονομασίας του αντικειμένου της εργασίας, της ομάδας στην οποία ανήκει και πολλές άλλες ενδιαφέρουσες πληροφορίες. Η JobDetails, με τη σειρά της, χρησιμοποιεί σαν αναφορά την org.quartz.JobDataMap. Η JobDataMap περιέχει στοιχεία για το χρήστη στον οποίο η δουλειά είναι καταχωρημένη.

Οι δουλειές και τα triggers χρονοπρογραμματίζονται μέσω της Quartz Scheduler διεπαφής. Για να πάρουμε ένα στιγμιότυπο ενός χρονοπρογραμματιστή πρέπει να το δημιουργήσουμε από το factory. Ανεξάρτητα από τον τύπο του χρονοπρογραμματιστή που χρησιμοποιείται, δεν πρέπει σε καμία περίπτωση να δημιουργείται μια υπόσταση του απευθείας, αλλά μόνο μέσω μιας μεθόδου του factory έτσι ώστε να εξακριβωθεί ότι όλα τα στοιχεία του έχουν αρχικοποιηθεί σωστά. Αυτό ακριβώς εξυπηρετεί η org.quartz.SchedulerFactory διεπαφή.

Ο πιο εύκολος τρόπος για να πραγματοποιηθεί κάτι τέτοιο είναι η κλήση της μεθόδου getDefaultScheduler() της κλάσης StdSchedulerFactory. Όταν χρησιμοποιείται το σύστημα του Quartz, είναι απαραίτητο να ξεκινήσει ο χρονοπρογραμματιστής μέσω της μεθόδου start(). Η πιο συνηθισμένη ακολουθία ενεργειών προκειμένου να χρησιμοποιηθεί ο Quartz σε μια εφαρμογή είναι:

- Η δημιουργία της εργασίας.
- Η δημιουργία και η απόδοση τιμών στα triggers.
- Ο χρονοπρογραμματισμός των εργασιών και των triggers που τρέχουν πάνω στον χρονοπρογραμματιστή.
- Η εκκίνηση του χρονοπρογραμματιστή.

Μια ακόμα άκρως ενδιαφέρουσα δυνατότητα που παρέχεται από το σύστημα του Quartz είναι ο χρονοπρογραμματισμός μιας εργασίας με τη βοήθεια ενός XML αρχείου. Το αρχείο περιέχει πληροφορίες σχετικές με τη εργασία και τα triggers, και διαβάζεται κατά την εκκίνηση της εφαρμογής. Το πλεονέκτημα αυτής της δυνατότητας είναι η ευκολία στην συντήρηση καθώς η οποιαδήποτε αλλαγή συνεπάγεται απλή αλλαγή ενός

XML αρχείου και εκκίνηση της εφαρμογής και όχι αλλαγές σε κώδικα, μεταγλώττιση και εγκατάσταση του νέου κώδικα. Ακολουθεί ένα ενδεικτικό παράδειγμα ενός τέτοιου XML αρχείου:

```
<?xml version='1.0' encoding='utf-8'?>
<quartz>
    <job>
        <job-detail>
            <name>testJob</name>
            <group>DEFAULT</group>
            <description>
                A job that scans an ftp site for files
            </description>
            <job-class>FTP</job-class>
            <job-data-map allows-transient-data="true">
                <entry>
                    <key>FTP_HOST</key>
                    <value>\home\cavaness\inbound</value>
                </entry>
             <!--  Other neccessary Job parameters here -->
            </job-data-map>
        </job-detail
        <trigger>
            <simple>
                <name>ScanFTPSiteJobTrigger</name>
                <group>DEFAULT</group>
                <job-name>ScanFTPSiteJob</job-name>
                <job-group>DEFAULT</job-group>
                <start-time>2005-09-11 6:10:00 PM</start-time>
                <!-- repeat indefinitely every 60 seconds -->
                <repeat-count>-1</repeat-count>
                <repeat-interval>60000</repeat-interval>
            </simple>
        </trigger>

    </job>
</quartz>
```

Σχήμα 1: XML αρχείο για το χρονοπρογραμματισμό μιας εργασίας.

Το σύστημα του Quartz διαθέτει πολλά ιδιαίτερα χαρακτηριστικά μεταξύ των οποίων ιδιαίτερη θέση έχουν οι Quartz Listeners. Οι Quartz Listeners είναι κλάσεις Java, που δημιουργούνται από τον χρήστη και μπορούν να κληθούν από το σύστημα του Quartz. Για παράδειγμα, όταν μια εργασία έχει χρονοπρογραμματιστεί ή όταν ένα trigger έχει ολοκληρωθεί, όλα αυτά τα γεγονότα μπορούν να προκαθοριστούν ώστε να ενημερώνουν ένα Listener. Το σύστημα διαθέτει Listeners για τον χρονοπρογραμματιστή, για τις δουλειές αλλά και για τα triggers, και είναι δυο ειδών, global ή σχετικοί μόνο με μια συγκεκριμένη εργασία ή με ένα συγκεκριμένο trigger.

Οι εφαρμογές που χρησιμοποιούν τον Quartz μπορούν να λειτουργήσουν και σε κατανεμημένο περιβάλλον, με τα ακόλουθα χαρακτηριστικά:

- Κλιμάκωση (scalability)
- Κατανομή φόρτου (load balancing)
- Υψηλή διαθεσιμότητα (high availability)
- Ανάκαμψη από σφάλματα (fail – over)

Η κάθε εργασία μπορεί να εκτελεστεί μέσα σε μια JTA συναλλαγή, απλά θέτοντας στο αρχείο quartz.properties, το οποίο αποτελεί την «καρδιά» του Quartz συστήματος, την παράμετρο org.quartz.scheduler.wrapJobExecutionInUserTransaction ίση με true. Με αυτή την επιλογή μια συναλλαγή θα ξεκινήσει προτού την κλήση της μεθόδου execute() και θα τερματιστεί μόλις η μέθοδος αυτή τερματιστεί.

Το quartz.properties αρχείο καθορίζει τη συμπεριφορά της Quartz εφαρμογής κατά την εκτέλεση, και περιέχει μια πληθώρα παραμέτρων οι οποίες καθορίζουν το πως συμπεριφέρεται ο Quartz, παρακάτω θα αναφερθούμε σε ορισμένες μόνο από τις βασικές. Στη συνέχεια, παραθέτουμε ένα αντιπροσωπευτικό παράδειγμα ενός βασικού quartz.properties αρχείου.

```
org.quartz.scheduler.instanceName = QuartzScheduler
org.quartz.scheduler.instanceId = AUTO
org.quartz.threadPool.threadCount = 5
org.quartz.threadPool.threadPriority = 5
org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
#============================================================
#Configure JobStore
#============================================================
org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore
#============================================================
#Configure Plugins
#============================================================
org.quartz.plugin.jobInitializer.class =
org.quartz.plugins.xml.JobInitializationPlugin
org.quartz.plugin.jobInitializer.overWriteExistingJobs = true
org.quartz.plugin.jobInitializer.failOnFileNotFound = true
org.quartz.plugin.jobInitializer.validating=false
```

Σχήμα 2: Παράδειγμα του Quartz properties αρχείου.

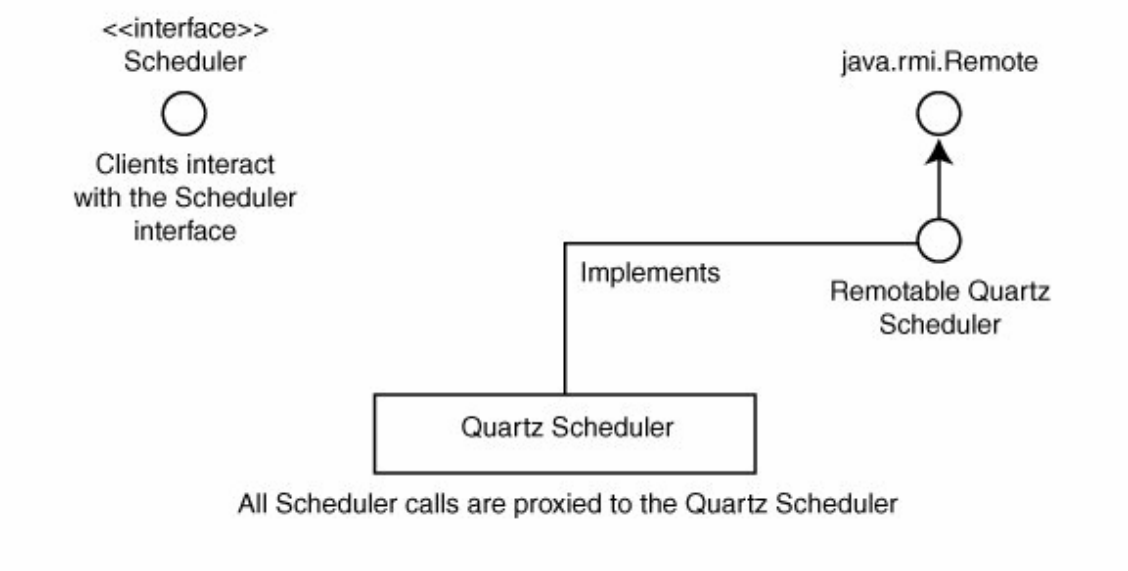
Το αρχείο αυτό διακρίνεται σε τέσσερα τμήματα, τα οποία διαχωρίζονται με βάση το πεδίο δράσης των παραμέτρων. Συγκεκριμένα, αποτελείται από τα εξής τμήματα:

- Scheduler Properties, αποτελείται από τις δυο πρώτες γραμμές, οι οποίες καθορίζουν το όνομα και το id για κάθε scheduler. Η τιμή της παραμέτρου org.quartz.scheduler.instanceName μπορεί να είναι οποιαδήποτε ακολουθία χαρακτήρων.
- Threadpool Properties, το τμήμα αυτό θέτει τις απαραίτητες τιμές για τα νήματα, τα οποία τρέχουν στο παρασκήνιο και είναι επιφορτισμένα με όλη την βαριά εργασία. Η παράμετρος threadCount καθορίζει πόσα worker threads θα δημιουργηθούν και θα πρέπει να έχει τιμή μεγαλύτερη ή ίση με 1. Η παράμετρος tHReadPriority καθορίζει την προτεραιότητα με την οποία τρέχουν τα νήματα, η μέγιστη τιμή είναι ίση με την σταθερά java.lang.Thread.MAX_PRIORITY, η οποία είναι ίση με 10, κα η ελάχιστη τιμή είναι ίση με την σταθερά java.lang.Thread.MIN_PRIORITY, η οποία είναι ίση με 1. Η τυπική τιμή για την παράμετρο είναι η Thread.NORM_PRIORITY, η οποία είναι ίση με 5.
- Jobstore Settings, οι παράμετροι αυτοί περιγράφουν το πως η πληροφορία για τις δουλειές και τα triggers αποθηκεύονται κατά την διάρκεια ζωής του scheduler.
- Plug-In Settings, το τελευταίο κομμάτι στο quartz.properties αρχείο είναι αυτό που καθορίζει οποιοδήποτε quartz plug-in που πρέπει να ρυθμιστεί.

3.2 Αρχιτεκτονική

Ο QuartzScheduler είναι η «καρδιά» του συστήματος και η μηχανή που το καθοδηγεί. Παρόλα αυτά, δεν περιλαμβάνεται όλη η λειτουργικότητα μέσα σε αυτό το συστατικό. Το σύστημα έχει σχεδιαστεί ώστε να είναι ευέλικτο και εύκολα ρυθμίσιμο ώστε πολλές από τις σημαντικές λειτουργίες να ρυθμίζονται από ανεξάρτητα κομμάτια και υποσυστήματα.

Στο σχήμα 3 παρουσιάζεται ένα διάγραμμα κλάσεων των βασικών τμημάτων του.



Σχήμα 3: Διάγραμμα βασικών τμημάτων του Quartz framework.

Το Quartz σύστημα περιλαμβάνει δυο βασικούς τύπους JobStore. Ο πρώτος τύπος, που χρησιμοποιεί τη μνήμη για να αποθηκεύει πληροφορίες χρονοπρογραμματισμού ονομάζεται RAMJobStore. Αυτός ο τρόπος είναι ο πιο εύκολος για να ρυθμιστεί, και για το μεγαλύτερο αριθμό εφαρμογών είναι αρκετός για να καλύψει τις ανάγκες σε διατήρηση αυτών των πληροφοριών. Όμως, επειδή η πληροφορία είναι αποθηκευμένη μόνο στο κομμάτι της μνήμης, κινδυνεύει να χαθεί όταν η εφαρμογή σταματήσει. Σε περιπτώσεις, κατά τις οποίες απαιτείται διατήρηση των πληροφοριών, πρέπει να χρησιμοποιηθεί ο δεύτερος τύπος JobStore.

Ο δεύτερος τύπος JobStore, ουσιαστικά προσφέρεται σε δυο υλοποιήσεις μέσα στο σύστημα, αλλά και οι δυο συνήθως καλούνται JDBCJobStore. Οι δυο τρόποι απαιτούν μια σχεσιακή βάση για την αποθήκευση των πληροφοριών και διαφέρουν στο αν οι συναλλαγές με τη βάση ελέγχονται από το χρήστη ή από τον εξυπηρετητή.

Οι δυο τύποι του JDBCJobStore είναι:

- JobStoreTX: ο οποίος χρησιμοποιείται όταν ο χρήστης ελέγχει τις συναλλαγές ή όταν χρησιμοποιείται μέσα σε έναν περιβάλλον έξω από κάποιον εξυπηρέτη εφαρμογών (application server).
- JobStoreCMT: όταν χρησιμοποιείται μέσα σε έναν εξυπηρέτη εφαρμογών και υπεύθυνος για τις συναλλαγές είναι ο J2EE container.

Οι δύο τύποι SchedulerFactory είναι ο org.quartz.impl.DirectSchedulerFactory και ο org.quartz.impl.StdSchedulerFactory. Το DirectSchedulerFactory προορίζεται για εκείνους τους χρήστες που επιθυμούν να έχουν πλήρη έλεγχο όσον αφορά τη λειτουργία του χρονοπρογραμματιστή. Όταν χρησιμοποιείται το DirectSchedulerFactory, τρία βασικά βήματα πρέπει να ακολουθηθούν:

- Δημιουργία ενός στιγμιότυπου από το factory χρησιμοποιώντας τη στατική μέθοδο getInstance().
- Αρχικοποίηση μέσω κλήσης μιας από τις μεθόδους της ομάδας createXXX, ανάλογα με το τύπο του χρονοπρογραμματιστή που θέλουμε να πάρουμε.
- Ανάκτηση ενός στιγμιότυπου από το factory χρησιμοποιώντας τη μέθοδο getScheduler().

Αντίθετα με το DirectSchedulerFactory, το org.quartz.impl.StdSchedulerFactory στηρίζεται σε ένα σύνολο από χαρακτηριστικά που καθορίζουν το πως θα κατασκευαστεί ένα στιγμιότυπο του χρονοδρομολογητή. Επιπλέον, μέσα από το Quartz σύστημα, μπορούν να γίνουν ενέργειες πάνω στην κατάσταση λειτουργίας του χρονοδρομολογητή, οι οποίες περιγράφονται παρακάτω:

- Έναρξη του χρονοδρομολογητή, μέσω της start() μεθόδου.
- Κατάσταση αναμονής, κατά την οποία ο χρονοδρομολογητής σταματάει να κοιτάει για δουλείες που αναμένουν εκτέλεση, ενέργεια χρήσιμη σε περίπτωση, για παράδειγμα, κατά την οποία είναι απαραίτητη η επανεκκίνηση της βάσης. Μπορεί να τεθεί σε κατάσταση αναμονής μέσω της μεθόδου standby().
- Τερματισμός του χρονοδρομολογητή.

Η πρώτη μέθοδος περιέχει μια παράμετρο που δηλώνει στο χρονοδρομολογητή, το αν θα περιμένει να ολοκληρωθούν οι προς εκτέλεση δουλειές ή όχι προτού τερματίσει την λειτουργία του.

Έχοντας αναφερθεί σε διάφορα χαρακτηριστικά ενός στιγμιότυπου του χρονοδρομολογητή, ας περιγράψουμε μερικά στοιχεία του αντικειμένου μιας εργασίας. Οι υποστάσεις δεν δημιουργούνται μέχρι να έρθει η στιγμή να εκτελεστούν. Κάθε φορά που μία εργασία εκτελείται, ένα καινούργιο στιγμιότυπο αυτής δημιουργείται. Βασικό πλεονέκτημα αυτής της διαδικασίας είναι ότι οι δουλειές δεν θα πρέπει να ανησυχούν

για την ασφάλεια των νημάτων καθώς ένα και μόνο ένα νήμα θα εκτελεί το δοσμένο instance ακόμα και στην περίπτωση που η ίδια εργασία εκτελείται ταυτόχρονα.

Η κατάσταση μιας εργασίας καθορίζεται μέσω του org.quartz.JobDataMap. Το JobDataMap υλοποιεί την java.util.Map μέσω της υπερκλάσης της org.quartz.utils.DirtyFlagMap. Στο JobDataMap υπάρχει η δυνατότητα αποθήκευσης ζευγαριών κλειδί / τιμή, ζευγάρια τα οποία είναι προσπελάσιμα από την κλάση της εργασίας. Ένα JobDataMap είναι διαθέσιμο στο επίπεδο των triggers το οποίο είναι ανάλογο με αυτό που περιγράφτηκε προηγουμένως με το επιπλέον στοιχείο ότι μπορεί να υποστηρίζει πολλαπλά triggers για την ίδια εργασία.

Αξίζει να αναφερθούμε σε ένα ακόμα χαρακτηριστικό του Quartz συστήματος, την υποστήριξη stateful και stateless δουλειών. Για την πρώτη ομάδα δουλειών παρέχεται η org.quartz.StatefulJob διεπαφή όταν απαιτείται η διατήρηση της κατάστασης μεταξύ των διαδοχικών εκτελέσεων της εργασίας. Δύο είναι οι κρίσιμες διαφορές μεταξύ μιας stateless και μιας stateful εργασίας:

- Η διατήρηση του αντικειμένου JobDataMap στο JobStore μετά από κάθε εκτέλεση, το οποίο διασφαλίζει το γεγονός ότι οι αλλαγές που γίνονται στο job data θα υπάρχουν στην επόμενη εκτέλεση.
- Η παράλληλη εκτέλεση δυο ή περισσοτέρων stateful JobDetail υποστάσεων. Για παράδειγμα, έχει δοθεί μια stateful JobDetail στο χρονοδρομολογητή και έχουν τεθεί δυο triggers για να πυροδοτήσουν την εργασία: ένα που την πυροδοτεί κάθε ένα λεπτό και ένα κάθε πέντε. Αν τα δυο triggers προσπαθήσουν να εκτελέσουν την εργασία την ίδια χρονική στιγμή, το σύστημα δεν θα το επιτρέψει να συμβεί. Το δεύτερο trigger θα μπλοκαριστεί μέχρι το πρώτο να ολοκληρώσει την εκτέλεση του.

Σαν τελευταία αναφορά σε επίπεδο δουλειών στο σύστημα του Quartz, θα περιγράψουμε τρεις ιδιότητες που μπορούν να δοθούν ή να μην δοθούν σε μια εργασία και τα οποία επηρεάζουν το περιβάλλον εκτέλεσης της. Ακολουθεί μια συνοπτική περιγραφή για κάθε μια από αυτές:

- Μια εργασία μπορεί να μην διατηρεί την κατάσταση της μεταξύ των τερματισμών των προγραμμάτων, μέσω της μεθόδου setVolatility(true) στην JobDetail.

- Μια εργασία μπορεί να διατηρείται στο JobStore, ακόμα και όταν δεν υπάρχουν άλλα triggers να την εκτελέσουν, γεγονός που την συνιστά διαθέσιμη για μελλοντικό χρονοπρογραμματισμό όταν σε κάποια χρονική στιγμή προστεθούν επιπλέον trigger για αυτή. Το αν θα διατηρείται μια εργασία ή όχι καθορίζεται από την μέθοδο setDurability() στην JobDetail (η προκαθορισμένη τιμή είναι false).
- Στην περίπτωση όπου η λειτουργία του χρονοδρομολογητή τερματιστεί αναπάντεχα και μια εργασία εκτελείται εκείνη τη στιγμή, η εργασία επανεκτελείται όταν ξεκινήσει πάλι η λειτουργία του χρονοδρομολογητή. Εφόσον, δεν υπάρχει τρόπος να γνωρίζει το σύστημα μέχρι ποιο σημείο είχε φτάσει η εκτέλεση της εργασίας όταν τερματίστηκε η λειτουργία του, η εργασία πρέπει να εκτελεστεί από την αρχή. Το αν θα επανεκτελείται η εργασία η όχι καθορίζεται από την μέθοδο setRequestsRecovery(boolean shouldRecover) στην JobDetail (η προκαθορισμένη τιμή είναι false).

Έχουμε ήδη διαπιστώσει πόσο εξαιρετικά σημαντικά είναι τα νήματα για τον Quartz καθώς είναι σχεδιασμένος με σκοπό να υποστηρίζει πολλές εκτελέσεις διεργασιών την ίδια στιγμή. Για να το πετύχει αυτό ο Quartz βασίζεται σημαντικά στη δυνατότητα της Java να εκμεταλλεύεται τη λειτουργία των νημάτων συνδυάζοντας και μερικές δικές του κλάσεις και διεπαφές. Όταν ένας Quartz δημιουργείται για πρώτη φορά μέσω της μεθόδου factory, η μέθοδος αυτή προετοιμάζει διάφορους πόρους που ο χρονοπρογραμματιστής πρόκειται να χρειαστεί κατά την διάρκεια ζωής του. Μερικοί από τους πόρους αυτούς έχουν να κάνουν με τα νήματα.

Όταν μία Quartz εφαρμογή ξεκινά για πρώτη φορά, το κύριο νήμα ξεκινά τον χρονοπρογραμματιστή. Ο QuartzScheduler δημιουργείται και με τη σειρά του δημιουργεί ένα στιγμιότυπο της org.quartz.core.QuartzSchedulerThread κλάσης. Η κλάση αυτή αναλαμβάνει την επεξεργασία με σκοπό να οριστεί πότε θα γίνει trigger η επόμενη εργασία. Όπως φαίνεται και από την ονομασία του το QuartzSchedulerThread είναι ένα νήμα. Η επεξεργασία που γίνεται στην κλάση αυτή φαίνεται παρακάτω:

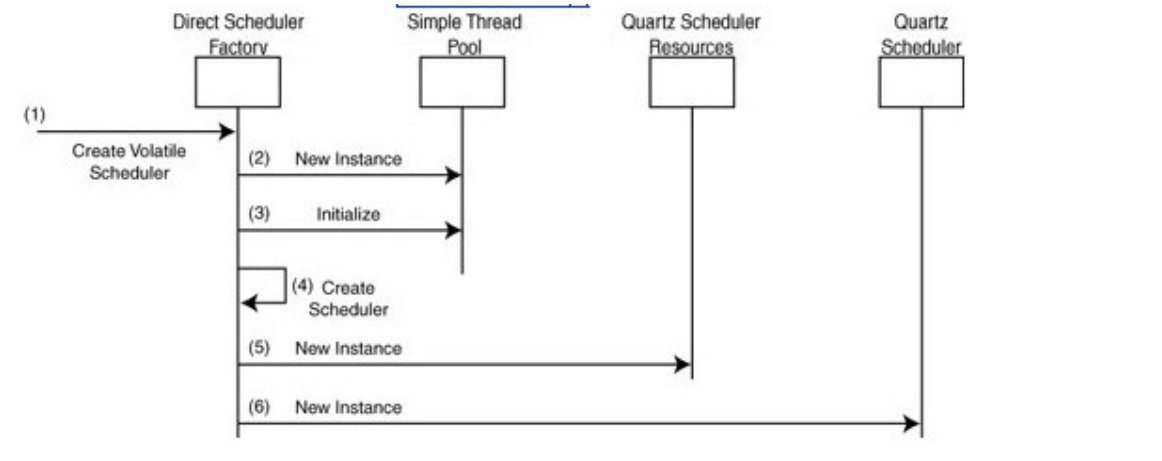
- Κατά την εκτέλεση του χρονοπρογραμματιστή:
    - Ελέγχει αν έχει ζητηθεί κατάσταση αναμονής. Σε περίπτωση που έχει ζητηθεί, περιμένει μέχρι να δεχθεί σήμα για να συνεχίσει.

- Ζητά από την JobStore το επόμενο trigger που θα εκτελέσει. Αν κανένα trigger δεν είναι έτοιμο προς εκτέλεση περιμένει για μικρό χρονικό διάστημα και ελέγχει ξανά.

- Αν υπάρχει trigger προς εκτέλεση, αναμένει για τον ακριβή χρόνο για να το πυροδοτήσει.
  - Όταν έρθει η κατάλληλη στιγμή, παίρνει το triggerFiredBundle για το αντίστοιχο trigger
  - Δημιουργεί ένα JobRunShell instance για την εργασία που χρησιμοποιεί το χρονοπρογραμματιστή και το triggerFiredBundle.
  - Ενημερώνει τη ThreadPool να εκτελέσει το JobRunShell όποτε είναι δυνατό.

Η παραπάνω λογική υπάρχει μέσα στη run() μέθοδο του QuartzSchedulerThread.

Όταν το Factory δημιουργεί ένα στιγμιότυπο του χρονοπρογραμματιστή, δίνει σε αυτόν και ένα στιγμιότυπο του QuartzSchedulerResources. Το QuartzSchedulerResources περιέχει, εκτός των άλλων, ένα ThreadPool αντικείμενο που παρέχει ένα pool από νήματα «εργάτες» υπεύθυνα για την εκτέλεση διεργασιών. Στον Quartz, ένα ThreadPool αντιπροσωπεύεται από τη org.quartz.spi.ThreadPool διεπαφή και περιέχει μία πλήρη υλοποίηση που καλείται org.quartz.simpl.SimpleThreadPool. Η SimpleThreadPool έχει έναν σταθερό αριθμό νημάτων «εργατών» που δεν μικραίνει ή μεγαλώνει ανάλογα με τον φόρτο. Στο σχήμα 4 φαίνεται η ακολουθία των βημάτων που εκτελούνται από το framework στην εκκίνηση του scheduler.



Σχήμα 4: Ακολουθία βασικών βημάτων κατά την εκκίνηση του scheduler.

Οι διεργασίες δεν επεξεργάζονται από τον Quartz στο κύριο νήμα. Αν αυτό γινόταν θα μείωνε κατά πολύ την ικανότητα κλιμάκωσης της εφαρμογής. Αντιθέτως, ο Quartz μεταβιβάζει το thread-management σε ένα ξεχωριστό κομμάτι του. Κατά το γενικό στήσιμο του Quartz, τα νήματα αναλαμβάνει η κλάση SimpleThreadPool. Η SimpleThreadPool δημιουργεί έναν αριθμό από WorkerThread υποστάσεις που είναι υπεύθυνες για την επεξεργασία κάθε εργασίας σε διαφορετικό

νήμα. Η WorkerThread είναι μία εσωτερική κλάση ορισμένη στη SimpleThreadPool κλάση και ουσιαστικά είναι ένα νήμα. Ο αριθμός αυτών των νημάτων αλλά και η προτεραιότητα τους ορίζεται στο αρχείο quartz.properties ή δίνονται στο factory.

Όταν η QuartzSchedulerThread ζητά από τη ThreadPool να εκτελέσει ένα JobRunShell αντικείμενο, η ThreadPool ελέγχει αν είναι διαθέσιμο κάποιο νήμα «εργάτης». Αν όλα τα νήματα είναι απασχολημένα, η ThreadPool περιμένει μέχρι κάποιο να ελευθερωθεί. Όταν κάποιο νήμα είναι διαθέσιμο και ένα JobRunShell αναμένει να εκτελεστεί, το νήμα «εργάτης» καλεί την run() μέθοδο της JobRunShell κλάσης.

Αν και τα νήματα «εργάτες» είναι ουσιαστικά Java νήματα, η κλάση JobRunShell γίνεται και αυτή εκτελέσιμη. Αυτό σημαίνει ότι λειτουργεί ως νήμα και περιέχει μία run() μέθοδο. Ο σκοπός της JobRunShell, όπως είδαμε παραπάνω, είναι να καλέσει την execute() μέθοδο σε μία εργασία. Εκτός αυτού, ενημερώνει την εργασία και τους trigger Listeners και τελειώνει ενημερώνοντας τις πληροφορίες των trigger για την εκτέλεση.

3.3 Προηγμένα χαρακτηριστικά

3.3.1 Quartz Cron Trigger

Στην πράξη, τα χρονοδιαγράμματα των εργασιών είναι πιο πολύπλοκα από τα triggers που έχουμε δει μέχρι στιγμής και σαφώς δεν περιορίζονται στα πλαίσια της εκτέλεσης μια εργασίας σε περιοδικό διάστημα. Αυτό τον σκοπό εξυπηρετούν τα Quartz Cron Triggers. Η μορφή του Quartz cron trigger μοιάζει με το UNIX cron με τις παρακάτω διαφορές:

- Η μορφή του Quartz επιτρέπει προγραμματισμό με λεπτομέρεια δευτερολέπτου, ενώ στην περίπτωση του UNIX cron αναφερόμαστε σε προσέγγιση λεπτού.
- Η μορφή του UNIX cron περιέχει πέντε πεδία (λεπτό, ώρα, μέρα, μήνας, ημέρα της εβδομάδας) ενώ του Quartz επτά.

Πίνακας 1: Τα πεδία του Quartz cron

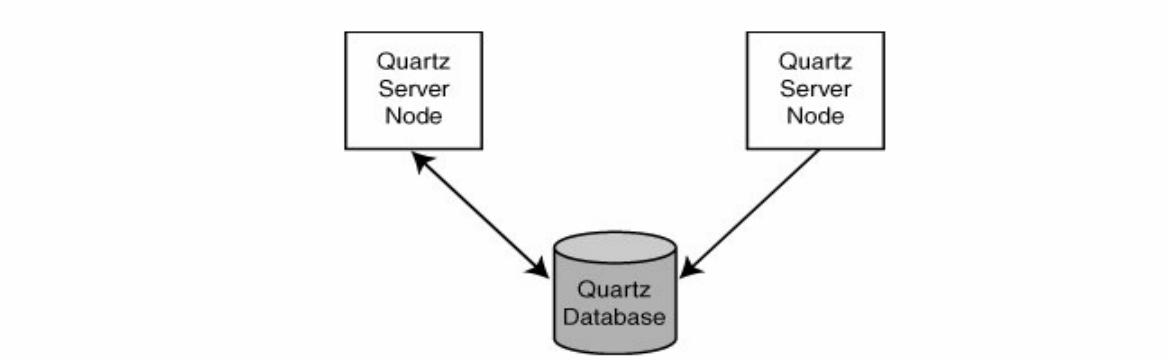
ΟΝΟΜΑ	ΑΠΑΙΤΕΙΤΑΙ	ΕΠΙΤΡΕΠΤΕΣ ΤΙΜΕΣ	ΕΙΔΙΚΟΙ ΧΑΡΑΚΤΗΡΕΣ
Δευτερόλεπτα	Ναι	059	, - * /
Λεπτά	Ναι	059	, - * /
Ώρες	Ναι	23	, - * /
Ημέρα του μήνα	Ναι	131	, - * ? / L W C
Μήνας	Ναι	112 ή JAN - DEC	, - * /
Ημέρα εβδομάδας	Ναι	17 ή SUN - SAT	, - * ? / L W #
Χρόνος	Όχι	Κενό ή 19702099	, - * /

Σε γενικές γραμμές, ο ειδικός χαρακτήρας '*' αντιστοιχεί σε όλες τις επιτρεπτές τιμές ενός πεδίου, ο χαρακτήρας '?' σε καμία τιμή του πεδίου, δηλαδή το πεδίο αυτό είναι ουσιαστικά αδιάφορο, ο χαρακτήρας ',' δηλώνει επιπρόσθετες τιμές για ένα πεδίο και οι χαρακτήρες '/' και '–' καθορίζουν προσαύξηση και εύρος, αντίστοιχα. Τέλος, ο χαρακτήρας 'L' αντιστοιχεί στην τελευταία επιτρεπτή τιμή ενός πεδίου, ο 'W' στις εργάσιμες μέρες και ο '#' σε συγκεκριμένη ημέρα ενός μήνα.

Οι cron εκφράσεις χρησιμοποιούνται για να καθοριστούν οι ημερομηνίες και οι ώρες που θα εκτελεστεί μια εργασία. Όταν δημιουργείται ένα CronTrigger στιγμιότυπο , αν δεν έχει οριστεί ο χρόνος έναρξης, το trigger θεωρεί ότι θα ξεκινήσει την εκτέλεση του με αφετηρία την ημερομηνία και ώρα που καθορίζεται από την cron έκφραση. Αν μια τέτοια συμπεριφορά δεν είναι η επιθυμητή τότε μπορούν να αξιοποιηθούν ο μέθοδοι setStartTime() και setEndTime() έτσι ώστε να καθοριστούν τα χρονικά πλαίσια μέσα στα οποία μια CronTrigger υπόσταση θα είναι ενεργή

3.3.2 Το Quartz framework σε clustered περιβάλλον

Κάθε κόμβος στον Quartz Enterprise Scheduler είναι μία ξεχωριστή εφαρμογή που λειτουργεί ανεξάρτητα από τους άλλους κόμβους. Αυτό σημαίνει ότι πρέπει κάθε κόμβος να ξεκινήσει και να σταματήσει ξεχωριστά. Αντίθετα με πολλούς εξυπηρετητές εφαρμογών που βρίσκονται σε συστοιχία, οι κόμβοι στον Quartz δεν επικοινωνούν μεταξύ τους ή με έναν κόμβο διαχειριστή. Οι Quartz εφαρμογές ενημερώνονται η μία για την ύπαρξη της άλλης μέσω των πινάκων της κοινής βάσης δεδομένων. Με βάση τα παραπάνω, εφόσον η επικοινωνία των κόμβων σε ένα περιβάλλον συστοιχίας βασίζεται εξ' ολοκλήρου στην ύπαρξη μίας κοινής βάσης δεν νοείται συστοιχία με τύπο JobStore διάφορο από JDBCJobStore, κατάσταση η οποία απεικονίζεται στο σχήμα 5.



Σχήμα 5: Επικοινωνία των υποστάσεων των χρονοπρογραμματιστων σε μια συστοιχία.

Ο Quartz Scheduler από μόνος του δεν έχει τη δυνατότητα να ανιχνεύει την ύπαρξη συστοιχίας, αλλά το JDBCJobStore μπορεί. Όταν ο Quartz χρονοδρομολογητής ξεκινά, καλεί τη μέθοδο schedulerStarted() η οποία, όπως γίνεται αντιληπτό και από το όνομά της, ενημερώνει το JobStore ότι ο χρονοδρομολογητής ξεκίνησε. Η schedulerStarted() μέθοδος υλοποιείται στην κλάση JobStoreSupport.

Η JobStoreSupport κλάση χρησιμοποιεί μια παράμετρο που τίθεται στο quartz.properties αρχείο και καθορίζει εάν η υπόσταση του χρονοδρομολογητή συμμετέχει σε μια συστοιχία. Εάν συμμετέχει, μια καινούργια υπόσταση του ClusterManager δημιουργείται και αρχικοποιείται. Η ClusterManager είναι μια εσωτερική κλάση της JobStoreSupport κλάσης, και επεκτείνει την java.lang.Thread. Η κλάση αυτή τρέχει περιοδικά και εκτελεί check-in συναρτήσεις. Όταν η clusterCheckin() μέθοδος

καλείται, η JobStoreSupport ανανεώνει το σχεσιακό πίνακα SCHEDULER_STATE για την υπόσταση του χρονοδρομολογητή. Ο χρονοδρομολογητής ελέγχει επίσης αν κάποιος από τους άλλους κόμβους έχει σταματήσει τη λειτουργία του.

Όταν μια υπόσταση ενός χρονοδρομολογητή εκτελεί τον check-in έλεγχο, ελέγχει αν υπάρχουν άλλες υποστάσεις που δεν εκτέλεσαν τον έλεγχο τη χρονική στιγμή που θα έπρεπε. Αυτό επιτυγχάνεται ελέγχοντας τον πίνακα SCHEDULER_STATE και ψάχνοντας για χρονοδρομολογητές που έχουν τιμή στο πεδίο LAST_CHECK_TIME παλιότερη από εκείνη που έχει τεθεί στην παράμετρο org.quartz.jobStore.clusterCheckinInterval.

Εάν ένας ή περισσότεροι κόμβοι δεν έχουν κάνει check-in, ο τρέχων χρονοδρομολογητής υποθέτει ότι αυτές οι υποστάσεις έχουν σταματήσει να λειτουργούν.

Όταν μια υπόσταση αποτύχει την ώρα που εκτελεί μία εργασία, είναι πιθανό η εργασία αυτή να εκτελεστεί από κάποιον από τους υπόλοιπους, τρέχοντες, χρονοδρομολογητές. Για να συμβεί αυτό, όπως έχουμε ήδη αναφέρει, πρέπει η παράμετρος που καθορίζει την ανάκτηση της εργασίας να τεθεί ως true στο αντικείμενο JobDetail. Αν αυτή η τιμή έχει τεθεί σε false τότε αν ένας χρονοδρομολογητής σταματήσει τη λειτουργία του ενώ μία εργασία τρέχει, αυτή δε θα επανεκτελεστεί. Το πόσο γρήγορα ανιχνεύεται η αποτυχία λειτουργίας ενός scheduler εξαρτάται από τη χρονική περίοδο που κάνει check-in, η τιμή της οποίας τίθεται στο αρχείο quartz.properties.

Τα βήματα για τη ρύθμιση του περιβάλλοντος της συστοιχίας στον Quartz είναι πολύ πιο εύκολα από το να ρυθμίζεις ένα παρόμοιο περιβάλλον σε μια J2EE συστοιχία. Τα βήματα αυτά είναι τα εξής:

- Ρύθμιση του quartz.properties αρχείου κάθε κόμβου.
- Ρύθμιση του JDBCJobStore.
- Φόρτωση της βάσης με πληροφορίες του χρονοδρομολογητή.
- Εκκίνηση κάθε κόμβου του Quartz χρονοδρομολογητή.

Σε ένα περιβάλλον συστοιχίας, κάθε κόμβος πρέπει να τροφοδοτηθεί με ένα quartz.properties αρχείο, κατάλληλα τροποποιημένο ώστε να επιτευχθεί η συστοιχία.

Ακολουθεί ένα παράδειγμα αυτού του αρχείου και η αντίστοιχη περιγραφή των παραμέτρων που συνδέονται με αυτή.

```
#===============================================================
#Configure Main Scheduler Properties
#===============================================================
org.quartz.scheduler.instanceName = TestScheduler1
org.quartz.scheduler.instanceId = AUTO
#===============================================================
#Configure ThreadPool
#===============================================================
org.quartz.threadPool.class = org.quartz.simpl.Simple ThreadPool
org.quartz.threadPool.threadCount = 5
org.quartz.threadPool.threadPriority = 5
org.quartz.jobStore.misfireThreshold = 60000
#===============================================================
#Configure JobStore
#===============================================================
org.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreTX
org.quartz.jobStore.driverDelegateClass =
org.quartz.impl.jdbcjobstore.MSSQLDelegate
org.quartz.jobStore.tablePrefix = QRTZ_
org.quartz.jobStore.dataSource = myDS

org.quartz.jobStore.isClustered = true
org.quartz.jobStore.clusterCheckinInterval = 20000

org.quartz.dataSource.myDS.driver = net.sourceforge.jtds.jdbc.Driver
org.quartz.dataSource.myDS.URL = jdbc:jtds:sqlserver:
//localhost:1433/quartz
org.quartz.dataSource.myDS.user = admin
org.quartz.dataSource.myDS.password = admin
org.quartz.dataSource.myDS.maxConnections = 10
```

Σχήμα 6: Παράδειγμα του Quartz properties αρχείου.

Όπως παρατηρούμε στο παραπάνω αρχείο, οι παράμετροι με την έντονη γραμματοσειρά είναι υπεύθυνες για την λειτουργία Quartz framework σε περιβάλλον συστοιχίας. Συγκεκριμένα, έχοντας επιλέξει σαν τύπο JobStore το JDBCJobStore, όπως έχει αναφερθεί παραπάνω είναι απαραίτητη η ύπαρξη μιας βάσης καθώς μόνο μέσω αυτής καθίσταται δυνατή η επικοινωνία των χρονοδρομολογητών στους επιμέρους κόμβους. Το επόμενο βήμα είναι η τροποποίηση των παραπάνω παραμέτρων και συγκεκριμένα:

- org.quartz.scheduler.instanceName, το όνομα της υπόστασης του χρονοδρομολογητή το οποίο, όπως έχουμε ήδη αναφέρει, μπορεί να είναι οποιαδήποτε ακολουθία χαρακτήρων.
- org.quartz.scheduler.instanceId, πρέπει να έχει υποχρεωτικά την τιμή auto.
- org.quartz.jobStore.isClustered, πρέπει να έχει υποχρεωτικά την τιμή true.

- org.quartz.jobStore.clusterCheckinInterval, καθορίζει τη συχνότητα (σε milliseconds) κατά την οποία ο χρονοδρομολογητής ελέγχει τη βάση ώστε να δει αν υπάρχουν άλλες υποστάσεις στη συστοιχία οι οποίες θα έπρεπε να είχαν δηλώσει την παρουσία τους κάνοντας check-in στη βάση και δεν το έχουν κάνει μέχρι κάποια συγκεκριμένη χρονική στιγμή. Με αυτό τον τρόπο, πραγματοποιείται η ανίχνευση των υποστάσεων στους άλλους κόμβους που έχουν σταματήσει την λειτουργία τους. Η προκαθορισμένη τιμή της είναι 15000.

Πρακτικά δεν υπάρχει καμία διαφορά στον τρόπο έναρξης ενός χρονοπρογραμματιστή σε περιβάλλοντα συστοιχίας και μη. Κάθε κόμβος πρέπει να ξεκινήσει την λειτουργία του ανεξάρτητα από τους άλλους. Κατά την εκκίνηση, μια υπόσταση συνδέεται με την βάση, παίρνει όλες τις απαραίτητες πληροφορίες για το χρονοπρογραμματιστή και ξεκινάει το χρονοπρογραμματισμό των εργασιών.

Ωστόσο, το Quartz σύστημα παρουσιάζει κάποιες αδυναμίες όσον αφορά μια συστοιχία. Μέχρι στιγμής, δεν υπάρχει η δυνατότητα της ανάθεσης μιας εργασίας σε ένα συγκεκριμένο κόμβο, αλλά ούτε του τρεξίματος μιας εργασίας σε κάθε κόμβο του cluster. Τέλος, το Quartz σύστημα υποστηρίζει οριζόντια συστοιχία, δηλαδή το τρέξιμο των κόμβων σε διαφορετικά μηχανήματα, αλλά και κάθετο, δηλαδή τρέξιμο των κόμβων στο ίδιο μηχάνημα.

ΚΕΦΑΛΑΙΟ 4

ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ

4.1 Κριτήρια και Μετρικές Αξιολόγησης

Οι μετρικές αξιολόγησης που μπορούν να χρησιμοποιηθούν όσον αφορά την απόδοση ενός χρονοπρογραμματιστή μπορούν να χωριστούν σε δυο κατηγορίες, σε μετρικές του συστήματος και σε μετρικές του χρήστη. Η πρώτη κατηγορία περιλαμβάνει το throughput, το utilization, το makespan και το efficacy. Τα δύο τελευταία χρησιμοποιούνται κυρίως σε κατανεμημένα υπολογιστικά περιβάλλοντα και για αυτό δεν ενδείκνυνται για τους σκοπούς που εξυπηρετεί η παρούσα μελέτη. Το utilization αποκλείεται καθώς προσπαθεί να συλλάβει πόσο αποτελεσματικά χρησιμοποιούνται οι πόροι του συστήματος και για αυτό το λόγο ταιριάζει περισσότερο σε χαμηλού επιπέδου χρονοδρομολογητές, όπως οι CPU χρονοδρομολογητές. Το throughput θα χρησιμοποιηθεί, καθώς δίνει μία αίσθηση για τις δυνατότητες του χρονοδρομολογητή. Η δεύτερη κατηγορία, η οποία είναι πρωτεύουσας σημασίας για ένα εξυπηρετητή εφαρμογών, περιλαμβάνει το χρόνο απόκρισης και τη μέση καθυστέρηση. Αυτές οι μετρικές εστιάζουν στη μέτρηση της απόδοσης από την πλευρά του τελικού χρήστη. Η περίληψη των παραπάνω μετρικών στο σύνολο των σεναρίων αξιολόγησης μπορεί να δώσει μία αρκετά καθαρή εικόνα των δυνατοτήτων και των ορίων του χρονοδρομολογητή.

Η πιο σημαντική μετρική σε ένα χρονοδρομολογητή πραγματικού χρόνου είναι ο πραγματικός χρόνος καθυστέρησης της εκτέλεσης μιας εργασίας. Η μετρική καθυστέρησης ορίζεται ως ο χρόνος από την πραγματοποίηση του trigger, μέχρι την εκκίνησης της εκτέλεσης μιας εργασίας. Στην περίπτωσή μας η καθυστέρηση περιέχει τόσο την καθυστέρηση του timer, όσο και το χρόνο που μια εργασία περιμένει στην ουρά. Ένα θετικό της καθυστέρησης είναι ότι δεν έχει να κάνει με τη φύση των προς εκτέλεση δουλειών (εκτός αν είναι ταξινομημένες σε κλάσεις προτεραιότητας). Έτσι ένα απλός αριθμητικός μέσος είναι αρκετό για να καταγράψει τις επιπτώσεις της μέσης καθυστέρησης. Αυτός ο χρόνος θα πρέπει να είναι μικρότερος από ένα δευτερόλεπτο για τη μεγάλη πλειοψηφία των εργασιών. Επομένως, η κατάλληλη μονάδα μέτρησης του χρόνου είναι τα milliseconds. Προκειμένου να μετρηθεί η μέση καθυστέρηση των διεργασιών και η συσχέτιση της καθυστέρησης με την κλιμάκωση του κάθε συστήματος

χρονοπρογραμματισμού, μια σειρά από σενάρια αξιολόγησης πραγματοποιήθηκαν, τα οποία θα αναλυθούν στην ενότητα που ακολουθεί.

Συνοψίζοντας μετριέται το throughput, μετρική η οποία δεν είναι τόσο σημαντική για τον τελικό χρήστη, αλλά σχετίζεται άμεσα με την κλιμάκωση του χρονοδρομολογητή και η τυπική απόκλιση (standard deviation) των καθυστερήσεων των διεργασιών.

4.2 Σενάρια Αξιολόγησης

Η αξιολόγηση της αποδοτικότητας των δύο scheduler frameworks πραγματοποιήθηκε σε μία συστοιχία αποτελούμενη από δύο υπολογιστές. Ο ένας υπολογιστής διέθετε μία AMD Athlon (tm) 64 Proccessor 3200+ 2,01 GHz CPU και ο δεύτερος μία Intel (R) Core (TM) 2 1,66 GHz. Το λειτουργικό σύστημα που χρησιμοποιήθηκε ήταν τα Windows XP Proffesional, ρυθμισμένο ώστε να εκτελείται το μικρότερο δυνατό σύνολο εργασιών στο παρασκήνιο. Ο J2EE-συμβατός εξυπηρετητής που χρησιμοποιήθηκε ήταν ο JBoss 4.0.4, και τα αποτελέσματα των μετρήσεων των σεναρίων αξιολόγησης αποθηκεύτηκαν σε μία MySQL 5.0 βάση. Ένα ακόμα κρίσιμο σημείο όσον αφορά την αξιολόγηση της απόδοσης είναι ο ακριβής συγχρονισμός των ρολογιών των δύο υπολογιστών. Για το λόγο αυτό χρησιμοποιήθηκε μία υπηρεσία συγχρονισμού ρολογιών.

Όσον αφορά την εκτέλεση των σεναρίων αξιολόγησης του Quartz χρονοδρομολογητή, αναπτύχθηκε μία web διεπαφή, χρησιμοποιώντας το εργαλείο ανάπτυξης λογισμικού NetBeans 5.5.

Για την παρούσα ανάλυση χρησιμοποιήθηκε μια πληθώρα σεναρίων αξιολόγησης που το καθένα από αυτά αντιστοιχούσε σε διαφορετικό φόρτο εργασίας. Ο κάθε φόρτος εργασίας (αιτήσεις για χρονοπρογραμματισμό) δημιουργήθηκε από το εργαλείο Apache JMeter. Το τελευταίο είναι μία Java εφαρμογή σχεδιασμένη για να φορτώνει σενάρια αξιολόγησης τόσο σε στατικούς όσο και σε δυναμικούς πόρους όπως ένα στατικό αρχείο, Java Servlet, Java Oblects, βάσεις, FTP εξυπηρετητές και πολλά ακόμα. Το Apache JMeter μπορεί να χρησιμοποιηθεί για να προσομειώσει μεγάλο φόρτο εργασίας σε ένα εξυπηρετητή, δίκτυο ή ένα αντικείμενο ώστε να αξιολογηθεί η δυνατότητα του ή να αναλυθεί η συνολική απόδοση υπό διαφορετικές συνθήκες φόρτου εργασίας.

Προκειμένου να αποκτήσουμε μια ακριβή εικόνα των διεργασιών που τρέχουν πίσω από το web tier του κάθε συστήματος χρονοπρογραμματισμού χρησιμοποιήθηκε ακόμα

ένα εργαλείο, το JProfiler 4.0.2. Το JProfiler δίνει τη δυνατότητα λειτουργίας του εξυπηρετητή μέσα από αυτό, απλά διευκρινίζοντας το μονοπάτι στο οποίο βρίσκεται το αντίστοιχο εκτελέσιμο αρχείο του εξυπηρετητή. Το εργαλείο αυτό αποδείχτηκε ιδανικό για την εξαγωγή ενός σαφούς συμπεράσματος όσον αφορά το χρόνο που καταναλίσκεται στις διάφορες συναλλαγές με τη βάση, καθώς παρέχει τη δυνατότητα καταγραφής του χρόνου που καταναλώνεται σε επίπεδο κλάσεων.

Τα σενάρια αξιολόγησης αναπτύχθηκαν με σκεπτικό να αντικατοπτρίζουν όχι μόνο ρεαλιστικές αλλά και ακραίες συνθήκες. Για το σκοπό αυτό, και οι δυο λύσεις χρονοπρογραμματισμού υποβλήθηκαν σε ποικίλλες τιμές παράλληλων αιτήσεων. Συγκεκριμένα, με τη βοήθεια του Apache JMeter, προσομοιώθηκε ο φόρτος εργασίας που προκαλείται από 1000, 2000, 3000 και 4000 παράλληλες αιτήσεις χρηστών σε επίπεδο milliseconds για χρονοπρογραμματισμό εργασιών. Η εργασία που χρησιμοποιήθηκε σε όλα τα σενάρια αξιολόγησης ήταν μια απλή Java κλάση, η οποία ενσωματώθηκε κατάλληλα στους δυο χρονοδρομολογητές.

Ένα ακόμα ενδιαφέρον σημείο για αξιολόγηση της συμπεριφοράς των δυο λύσεων χρονοπρογραμματισμού, ήταν η εκπόνηση σεναρίων αξιολόγησης που να διαφοροποιούνται σε επίπεδο τιμών περιόδου. Προκειμένου να διερευνηθεί το κατά πόσον οι διαφορετικές τιμές της περιόδου και σε ποιο βαθμό επηρεάζουν τη συμπεριφορά των δυο συστημάτων επιλέχθηκαν ποικίλες τιμές ώστε να καλύπτουν ένα ευρύ φάσμα. Συγκεκριμένα, οι τιμές που δοκιμάστηκαν ήταν 60000, 90000, 180000, 220000 και 300000 milliseconds. Τα σενάρια των διαφορετικών τιμών περιόδου πραγματοποιήθηκαν μόνο για την περίπτωση των 1000 παράλληλων αιτήσεων χρηστών, καθώς οι αντίστοιχες μετρήσεις για μεγαλύτερο φόρτο εργασίας δεν εξυπηρετούν τον αρχικό σκοπό ανάλυσης της επίδρασης της περιόδου.

Όπως έχει αναφερθεί στο κεφάλαιο 2, το ιδιαίτερο χαρακτηριστικό της αρχιτεκτονικής του συστήματος του Polos Enterprise Scheduler είναι η αξιοποίηση της JBoss Cache, σε επίπεδο τέτοιο ώστε να αποτελεί την «καρδιά» του συστήματος.

Η JBossCache μπορεί να ρυθμιστεί είτε να λειτουργεί με σύγχρονη αντιγραφή (synchronous replication) είτε με ασύγχρονη αντιγραφή (asynchronous replication). Στην πρώτη περίπτωση, κάθε προσπάθεια για τοποθέτηση ενός νέου μηνύματος «αναχαιτίζεται» μέχρι όλες οι αλλαγές να έχουν μεταφερθεί επιτυχώς σε όλους τους κόμβους της συστοιχίας. Ενώ, στην δεύτερη περίπτωση, η όλη διαδικασία εκτελείται στο

παρασκήνιο, μη επηρεάζοντας τη λειτουργία των κόμβων σε τόσο μεγάλο βαθμό όσο η πρώτη περίπτωση. Επομένως, ένα ακόμα ενδιαφέρον σημείο, είναι η ανάλυση της συμπεριφοράς του Polos Enterprise Scheduler όταν η JBoss Cache παρουσιάζει διαφορετικό τύπο αντιγραφής. Για το σκοπό αυτό, τα σενάρια αξιολόγησης που δημιουργήθηκαν εκτελέστηκαν και για τους δυο τύπους.

Ακολουθούν πίνακες τιμών μόνο των παραμέτρων χρονοπρογραμματισμού για κάθε ένα από τα σενάρια αξιολόγησης, τα οποία εκτελέστηκαν για:

- Polos Enterprise Scheduler με σύγχρονη αντιγραφή.
- Polos Enterprise Scheduler με ασύγχρονη αντιγραφή.
- Quartz Enterprise Scheduler.

Πίνακας 1: Σενάριο Αξιολόγησης 1

Polos Enterprise Scheduler – Quartz Enterprise Scheduler
1000 ΤΑΥΤΟΧΡΟΝΕΣ ΑΙΤΗΣΕΙΣ ΧΡΗΣΤΩΝ
URL Παράμετροι
Start: 0 ms **Period**: 60000 ms **Repetitions**: 10

Πίνακας 2: Σενάριο Αξιολόγησης 2

Polos Enterprise Scheduler – Quartz Enterprise Scheduler
1000 ΤΑΥΤΟΧΡΟΝΕΣ ΑΙΤΗΣΕΙΣ ΧΡΗΣΤΩΝ
URL Παράμετροι
Start: 0 ms **Period**: 90000 ms **Repetitons**: 10

Πίνακας 3: Σενάριο Αξιολόγησης 3

Polos Enterprise Scheduler – Quartz Enterprise Scheduler
1000 ΤΑΥΤΟΧΡΟΝΕΣ ΑΙΤΗΣΕΙΣ ΧΡΗΣΤΩΝ
URL Παράμετροι
Start: 0 ms **Period**: 180000 ms **Repetitions**: 10

Πίνακας 4: Σενάριο Αξιολόγησης 4

Polos Enterprise Scheduler – Quartz Enterprise Scheduler
1000 ΤΑΥΤΟΧΡΟΝΕΣ ΑΙΤΗΣΕΙΣ ΧΡΗΣΤΩΝ
URL Παράμετροι
Start: 0 ms **Period**: 220000 ms **Repetitions**: 10

Πίνακας 5: Σενάριο Αξιολόγησης 5

Polos Enterprise Scheduler – Quartz Enterprise Scheduler
1000 ΤΑΥΤΟΧΡΟΝΕΣ ΑΙΤΗΣΕΙΣ ΧΡΗΣΤΩΝ
URL Παράμετροι
Start: 0 ms **Period**: 300000 ms **Repetitions**: 10

Πίνακας 6: Σενάριο Αξιολόγησης 6

Polos Enterprise Scheduler – Quartz Enterprise Scheduler
2000 ΤΑΥΤΟΧΡΟΝΕΣ ΑΙΤΗΣΕΙΣ ΧΡΗΣΤΩΝ
URL Παράμετροι
Start: 0 ms **Period**: 60000 ms **Repetitions**: 10

Πίνακας 7: Σενάριο Αξιολόγησης 7

Polos Enterprise Scheduler – Quartz Enterprise Scheduler
3000 ΤΑΥΤΟΧΡΟΝΕΣ ΑΙΤΗΣΕΙΣ ΧΡΗΣΤΩΝ
URL Παράμετροι
Start: 0 ms **Period**: 60000 ms **Repetitions**: 10

Πίνακας 8: Σενάριο Αξιολόγησης 8

Polos Enterprise Scheduler – Quartz Enterprise Scheduler
4000 ΤΑΥΤΟΧΡΟΝΕΣ ΑΙΤΗΣΕΙΣ ΧΡΗΣΤΩΝ
URL Παράμετροι
Start: 0 ms **Period**: 60000 ms **Repetitions**: 10

ΚΕΦΑΛΑΙΟ 5

ΣΤΑΤΙΣΤΙΚΗ ΑΝΑΛΥΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΑΞΙΟΛΟΓΗΣΗΣ

5.1 Ανάλυση επιδόσεων Polos Enterprise Scheduler

Για την ανάλυση επιδόσεων των δυο λύσεων χρονοπρογραμματισμού, που αποτελούν αντικείμενο έρευνας της παρούσας μελέτης, πραγματοποιήθηκε μια σειρά από σενάρια αξιολόγησης, καθένα από τα οποία αντιστοιχούν σε συνθήκες φόρτου εργασίας, οι οποίες αναλυτικά παρουσιάστηκαν στο προηγούμενο κεφάλαιο. Όπως έχει ήδη αναφερθεί, ο αντικειμενικός σκοπός της διεξαγωγής αυτών των σεναρίων αξιολόγησης δεν είναι μόνο η ανάλυση των επιδόσεων των δυο λύσεων χρονοπρογραμματισμού αλλά και ο προσδιορισμός της επίδρασης της περιόδου εκτέλεσης των εργασιών στην επίδοση του καθενός.

Στην παρούσα ενότητα, θα γίνει μια προσπάθεια να εκτιμηθεί η επίδοση του Polos Enterprise Scheduler μέσω της παρουσίασης των αποτελεσμάτων για κάθε ένα από τα σενάρια αξιολόγησης. Συγκεκριμένα, για κάθε σενάριο αξιολόγησης αντιστοιχεί ένας πίνακας αποτελεσμάτων, στον οποίο καταγράφονται οι εξής τιμές:

- Η αναμενόμενη διάρκεια εκτέλεσης όλων των χρονοπρογραμματιζόμενων εργασιών.
- Η τυπική απόκλιση της ολικής καθυστέρησης των εργασιών.
- Η μέση καθυστέρηση της κάθε εργασίας από τη χρονική στιγμή που πυροδοτείται μέχρι να γίνει trigger (T1).
- Το χρονικό διάστημα που η κάθε εργασία παραμένει στην ουρά αναμονής (T2).
- Το χρονικό διάστημα για μια εργασία από τη στιγμή που έφυγε από την ουρά μέχρι να εκτελεστεί (T3).

Η ανάλυση των επιδόσεων του Polos Enterprise Scheduler πραγματοποιείται με βάση δυο βασικούς άξονες, τον διαφορετικό φόρτο εργασίας και την διαφορετική περίοδο των χρονοπρογραμματιζόμενων εργασιών. Για αυτό και τα αποτελέσματα παρουσιάζονται και αναλύονται σε ξεχωριστές υποενότητες. Όπως έχει αναφερθεί, για την περίπτωση του Polos Enterprise Scheduler, ιδιαίτερο ενδιαφέρον παρουσιάζει η ανάλυση της συμπεριφοράς του όταν η JBoss Cache παρουσιάζει διαφορετικό τύπο

αντιγραφής (asynchronous, synchronous). Για το σκοπό αυτό, τα σενάρια αξιολόγησης που δημιουργήθηκαν εκτελέστηκαν και για τους δυο τύπους και οι αντίστοιχοι πίνακες αποτελεσμάτων διακρίνονται και με βάση αυτό το κριτήριο.

Επίδραση του διαφορετικού φόρτου εργασίας στην απόδοση:

Ακολουθούν οι πίνακες αποτελεσμάτων για τα σενάρια αξιολόγησης με φόρτο εργασίας 1000, 2000, 3000 και 4000 παράλληλων αιτήσεων χρηστών και με σταθερή περίοδο 60000 ms μεταξύ των επαναλήψεων.

Polos Enterprise Scheduler με ασύγχρονη αντιγραφή:

Πίνακας 1: Σενάριο Αξιολόγησης 1

Αναμενόμενη διάρκεια εκτέλεσης:			660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
197,50 ms	433,8 ms	0,75 ms	50,3 ms

Πίνακας 2: Σενάριο Αξιολόγησης 6

Αναμενόμενη διάρκεια εκτέλεσης:			660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
254,60 ms	490,77 ms	0,91 ms	133,6 ms

Πίνακας 3: Σενάριο Αξιολόγησης 7

Αναμενόμενη διάρκεια εκτέλεσης:			660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
5447,10 ms	1197,805 ms	1,935 ms	10679,638 ms

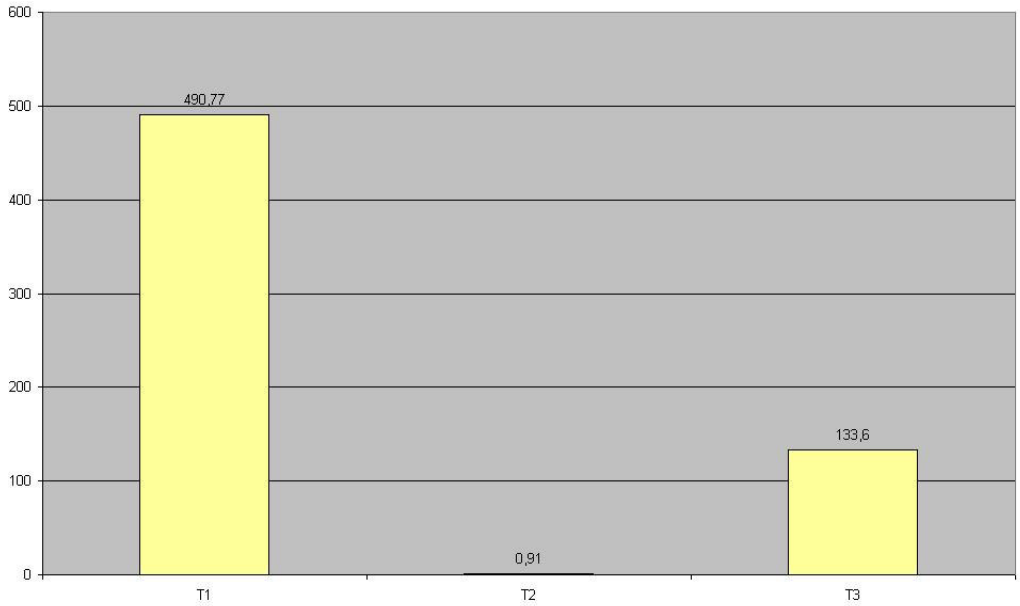
Πίνακας 4: Σενάριο Αξιολόγησης 8

Αναμενόμενη διάρκεια εκτέλεσης:			660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
80836,30 ms	65902,188 ms	4,246 ms	66232,961 ms

Οι παραπάνω πίνακες τιμών αντιστοιχούν σε φόρτο εργασίας που αυξάνεται κατά 1000 παράλληλες αιτήσεις κάθε φορά. Αναλύοντας τις τιμές της τυπικής απόκλισης της ολικής καθυστέρησης των εργασιών για κάθε ένα από τα σενάρια αξιολόγησης, παρατηρούμε ότι αυξάνεται με ρυθμό ανάλογο του φόρτου εργασίας. Για το πρώτο σενάριο αξιολόγησης, των 1000 παράλληλων αιτήσεων χρηστών, η τυπική απόκλιση είναι 197,5 ms, και αυξάνεται σε 254,6 ms, 5447,1 ms και 80836,3 ms για κάθε επιπρόσθετες 1000 παράλληλες αιτήσεις. Ιδιαίτερο ενδιαφέρον στην ανάλυση μας, παρουσιάζει ο προσδιορισμός των επιμέρους καθυστερήσεων T1, T2, και T3 ,οι οποίες έχουν οριστεί παραπάνω, και συνοπτικά αντιστοιχούν:

- στη μέση καθυστέρηση της κάθε εργασίας μέχρι να γίνει trigger,
- στην καθυστέρηση λόγω παραμονής στην ουρά αναμονής,
- στην καθυστέρηση μεταξύ εξόδου από την ουρά και εκτέλεσης.

Όπως παρατηρούμε από τις παραπάνω τιμές στα πεδία αυτά, οι τιμές των καθυστερήσεων T1 και T3 κυμαίνονται σε μεγαλύτερη κλίμακα συγκριτικά με τις τιμές της καθυστέρησης T2. Ακολουθεί ένα διάγραμμα που παρουσιάζει τις τιμές των καθυστερήσεων, το διάγραμμα αυτό δημιουργήθηκε με βάση τις τιμές του έκτου σεναρίου αξιολόγησης το οποίο αντιστοιχεί σε 2000 ταυτόχρονες αιτήσεις και δείχνει το πως διαμορφώνεται η ολική καθυστέρηση, T αλλά και οι επιμέρους καθυστερήσεις T1, T2 και T3 για το συγκεκριμένο σενάριο αξιολόγησης. Για την απεικόνιση αυτή επιλέχθηκαν οι τιμές του πίνακα 2, καθώς ο φόρτος των 2000 αιτήσεων αντιστοιχεί στο μέσο φόρτο των ποικίλων συνθηκών που προσομοιώθηκαν.



Διάγραμμα 1: T1, T2, T3 καθυστερήσεις. Ασύγχρονη αντιγραφή.

Polos Enterprise Scheduler με σύγχρονη αντιγραφή:

Πίνακας 5: Σενάριο Αξιολόγησης 1

Αναμενόμενη διάρκεια εκτέλεσης:			660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
22916 ms	53304,94 ms	3,246 ms	72,788 ms

Πίνακας 6: Σενάριο Αξιολόγησης 6

Αναμενόμενη διάρκεια εκτέλεσης:			660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
29195 ms	41549,09 ms	7,077 ms	81,168 ms

Πίνακας 7: Σενάριο Αξιολόγησης 7

Αναμενόμενη διάρκεια εκτέλεσης:			660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
28835,60 ms	44813,371 ms	15,24 ms	7849,984 ms

Πίνακας 8: Σενάριο Αξιολόγησης 8

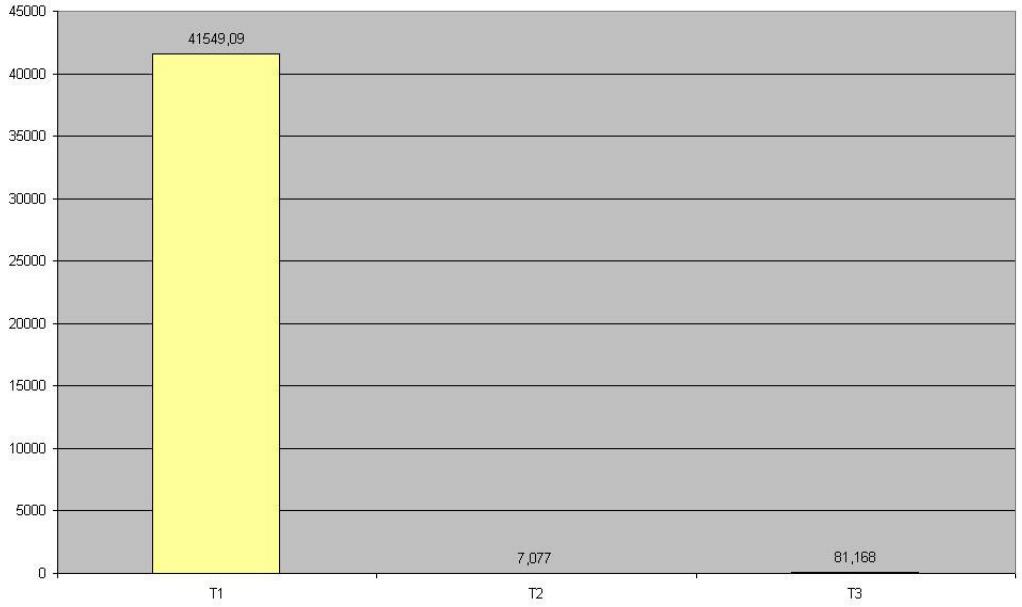
Αναμενόμενη διάρκεια εκτέλεσης:			660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
80532,10 ms	87810,508 ms	35,766 ms	58938,169 ms

Για την περίπτωση του Polos Enterprise Scheduler με σύγχρονη αντιγραφή, τα αποτελέσματα για τα αντίστοιχα σενάρια αξιολόγησης με τον τύπο της σύγχρονης αντιγραφής παρουσιάζουν διαφορετική συμπεριφορά, η οποία εντοπίζεται στα διαφορετικά επίπεδα διακύμανσης των επιμέρους καθυστερήσεων. Στην περίπτωση της σύγχρονης αντιγραφής, οι παρατηρούμενες τιμές των καθυστερήσεων είναι μικρότερες από τις αντίστοιχες τιμές στην περίπτωση της ασύγχρονης αντιγραφής. Η μέση καθυστέρηση της κάθε εργασίας μέχρι να γίνει trigger (T1) είναι αυτή που παρουσιάζει τη μέγιστη διαφοροποίηση μεταξύ των αντίστοιχων σεναρίων που εκτελέστηκαν.

Επίσης, αξιοσημείωτη διαφορά ανάμεσα στους δυο τύπους αντιγραφής παρατηρείται στις τιμές της τυπικής απόκλισης. Τα αποτελέσματα δείχνουν ότι για σενάρια αξιολόγησης με ήπιο φόρτο εργασίας (μέχρι 3000 παράλληλες αιτήσεις χρηστών) η τελευταία έχει σαφώς μικρότερες τιμές στην περίπτωση της ασύγχρονης αντιγραφής και ακολούθως η απόδοση του χρονοδρομολογητή προσεγγίζει περισσότερο την αναμενόμενη. Ενώ, στα σενάρια αξιολόγησης με πιο έντονο φόρτο εργασίας (4000 παράλληλες αιτήσεις χρηστών και παραπάνω), η τυπική απόκλιση αυξάνει σημαντικά και οι δυο τύποι αντιγραφής παρουσιάζουν σχεδόν ίδιες τιμές.

Ακολουθεί διάγραμμα των τιμών της ολικής καθυστέρησης T και των καθυστερήσεων T1, T2 και T3. Το διάγραμμα αυτό δημιουργήθηκε με βάση τις τιμές του έκτου πίνακα σε

αναλογία με το αντίστοιχο διάγραμμα που παρουσιάστηκε στην περίπτωση του Polos Enterprise Scheduler με ασύγχρονη αντιγραφή.



Διάγραμμα 2: T1, T2, T3 καθυστερήσεις. Σύγχρονη αντιγραφή.

Επίδραση της περιόδου στην απόδοση:

Polos Enterprise Scheduler με ασύγχρονη αντιγραφή:

Πίνακας 9: Σενάριο Αξιολόγησης 2

Αναμενόμενη διάρκεια εκτέλεσης:			930000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
266,60 ms	462,973 ms	0,7168 ms	81,583 ms

Πίνακας 10: Σενάριο Αξιολόγησης 3

Αναμενόμενη διάρκεια εκτέλεσης:			1740000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
282,40 ms	419,236 ms	0,628 ms	369,3513 ms

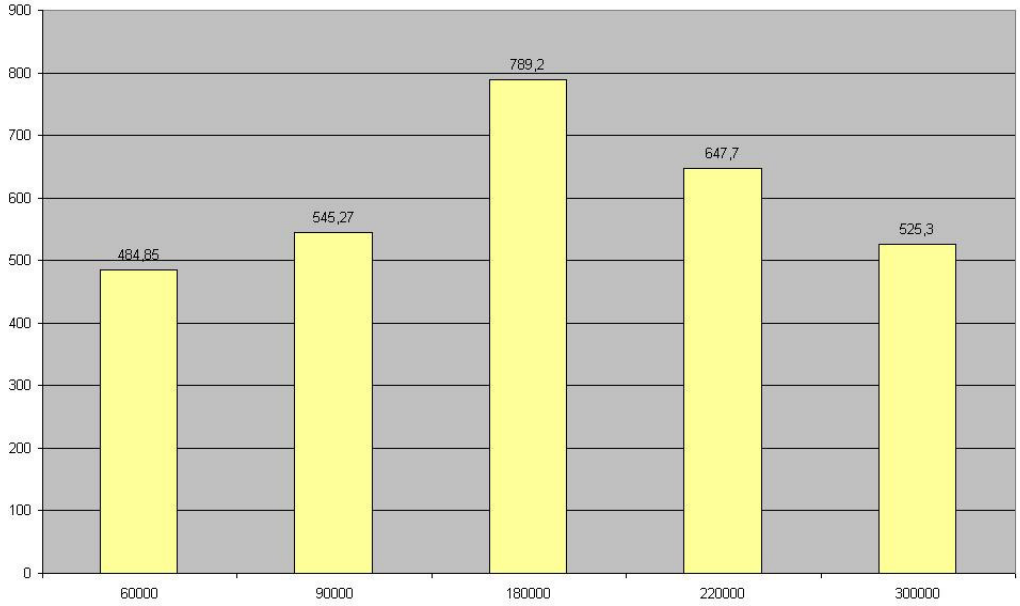
Πίνακας 11: Σενάριο Αξιολόγησης 4

Αναμενόμενη διάρκεια εκτέλεσης:			1980000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
238,30 ms	420,896 ms	0,704 ms	226,116 ms

Πίνακας 12: Σενάριο Αξιολόγησης 5

Αναμενόμενη διάρκεια εκτέλεσης:			2700000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**	**T3**
177,04 ms	413,92 ms	0,547 ms	110,841 ms

Ένας επιπλέον σκοπός των σεναρίων αξιολόγησης, όπως έχει ήδη ειπωθεί, είναι ο προσδιορισμός της επίδρασης των διαφορετικών τιμών της περιόδου στην απόδοση των δυο λύσεων χρονοπρογραμματισμού. Τα παραπάνω αποτελέσματα, υποδηλώνουν ότι για επιλεγμένες διακυμάνσεις της περιόδου που χρησιμοποιήθηκαν στα σενάρια αξιολόγησης, η απόδοση του χρονοδρομολογητή δεν παρουσιάζει αξιοσημείωτη βελτίωση. Οι τιμές των ολικών καθυστερήσεων για τις διάφορες τιμές περιόδου (60000, 90000, 180000, 220000 και 300000 ms) περιγράφονται στο διάγραμμα 3.



Διάγραμμα 3: Η διακύμανση της ολικής καθυστέρησης για τις διάφορες τιμές περιόδου

5.2 Ανάλυση επιδόσεων Quartz Enterprise Scheduler

Για την περίπτωση του Quartz Enterprise Scheduler, στους πίνακες των αποτελεσμάτων περιλαμβάνονται οι εξής τιμές:

- Η αναμενόμενη διάρκεια εκτέλεσης όλων των χρονοπρογραμματιζόμενων εργασιών.
- Η τυπική απόκλιση της ολικής καθυστέρησης των διεργασιών.
- Η μέση καθυστέρηση της κάθε εργασίας από τη χρονική στιγμή που πυροδοτείτε μέχρι να γίνει trigger (T1).
- Το χρονικό διάστημα κατά μέσο όρο που δαπανάται για τη κάθε εργασία στις οποιεσδήποτε αλληλεπιδράσεις με τη βάση ώστε να επιτευχθεί ο συγχρονισμός της κατά τη διάρκεια των διαφόρων εκτελέσεων των εργασιών (T2).

Στην αρχιτεκτονική του Quartz Scheduler δεν περιλαμβάνεται ουρά μηνυμάτων, επομένως η καθυστέρηση T3, η οποία στην περίπτωση του Polos Scheduler αντιστοιχούσε στο χρονικό διάστημα από τη στιγμή που μια εργασία έφυγε από την ουρά μέχρι να εκτελεστεί, δεν έχει νόημα να μετρηθεί.

Επίδραση του διαφορετικού φόρτου εργασίας στην απόδοση:

Ακολουθούν οι πίνακες που αντιστοιχούν στα σενάρια αξιολόγησης για φόρτο εργασίας 1000, 2000, 3000 και 4000 παράλληλων αιτήσεων χρηστών, με σταθερή περίοδο 60000 ms μεταξύ των επαναλήψεων.

Πίνακας 13: Σενάριο Αξιολόγησης 1

Αναμενόμενη διάρκεια εκτέλεσης:		660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**
697113,30 ms	135,79 ms	91905 ms

Πίνακας 14: Σενάριο Αξιολόγησης 6

Αναμενόμενη διάρκεια εκτέλεσης:		660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**
1392472,60 ms	120,6 ms	198556 ms

Πίνακας 15: Σενάριο Αξιολόγησης 7

Αναμενόμενη διάρκεια εκτέλεσης:		660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**
2299500,13 ms	128,7 ms	225517 ms

Πίνακας 16: Σενάριο Αξιολόγησης 8

Αναμενόμενη διάρκεια εκτέλεσης:		660000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**
3253048,50 ms	134,34 ms	375931 ms

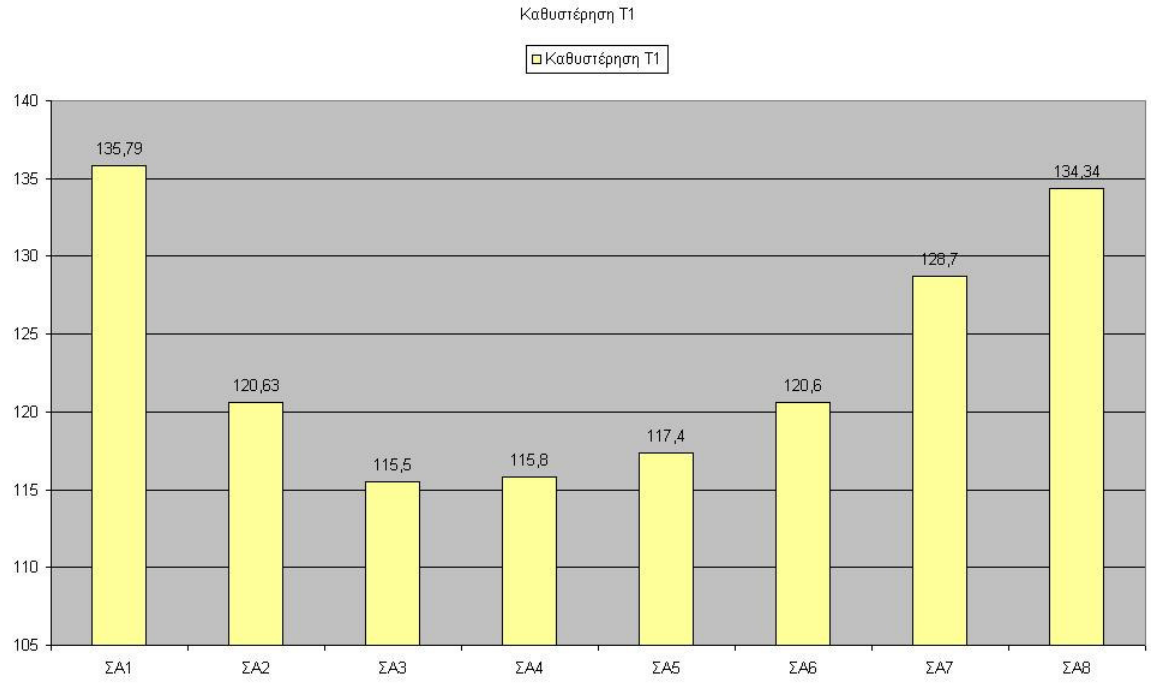
Αναλύοντας τα αποτελέσματα των παραπάνω πινάκων, καταλήγουμε σε συμπεράσματα ανάλογα με εκείνα που ισχύουν στην περίπτωση του Polos Enterprise Scheduler, όσων αφορά τους παράγοντες με τους οποίους αξιολογείται η απόδοση των δυο συστημάτων. Συγκεκριμένα, για κάθε αύξηση του φόρτου εργασίας κατά 1000 παράλληλες αιτήσεις χρηστών, δηλαδή για 1000, 2000, 3000 και 4000 αιτήσεις, η τιμή της τυπικής απόκλισης αυξάνεται, ξεκινώντας από 697113,30 ms, σε 1392472,60 ms, 2299500,13 ms και καταλήγοντας σε 3253048,50 ms. Η διακύμανση της τιμής της τυπικής απόκλισης αντικατοπτρίζει την επίδραση της αύξησης του φόρτου εργασίας στην επίδοση του συστήματος, δηλαδή όσο ο φόρτος εργασίας αυξάνει, η τιμή της τυπικής απόκλισης αυξάνει και συνεπώς επηρεάζεται αρνητικά η απόδοση του όλου συστήματος. Η περιγραφείσα συμπεριφορά βρίσκεται σε πλήρη αναλογία με την συμπεριφορά του Polos Enterprise Scheduler.

Αντίστοιχα με την περίπτωση του Polos Enterprise Scheduler, μετρήθηκαν οι εξής δυο καθυστερήσεις:

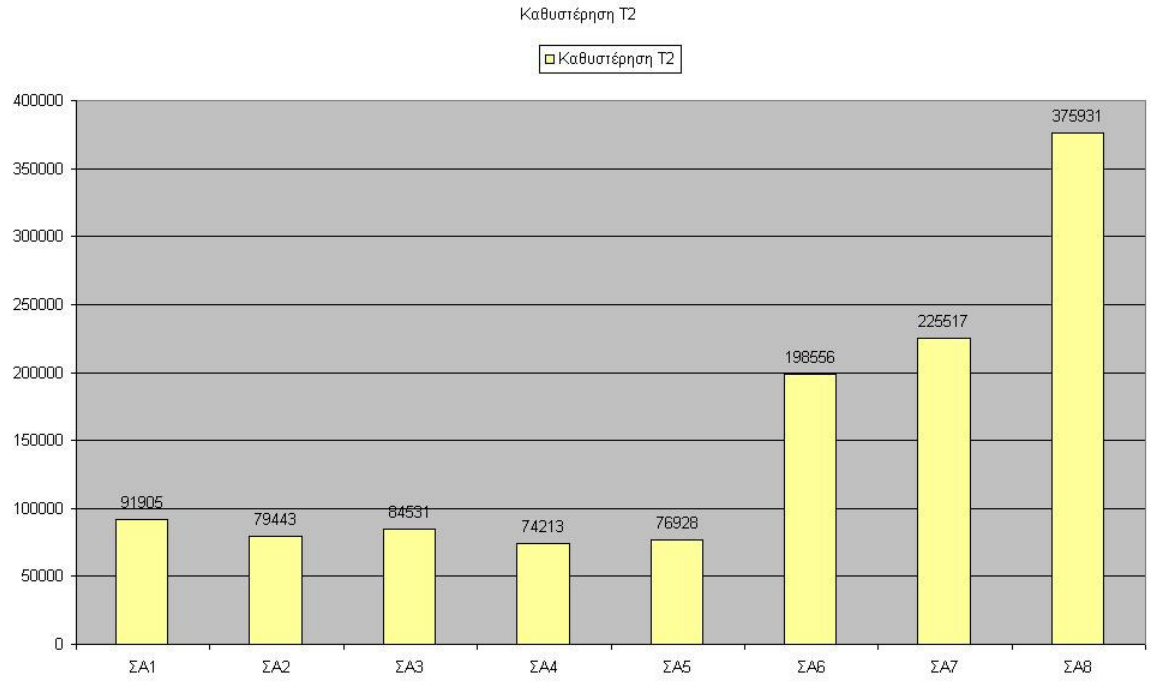
- η καθυστέρηση που μεσολαβεί μέχρι να γίνει trigger μια εργασία,
- η καθυστέρηση που οφείλεται στις αλληλεπιδράσεις με μια βάση ώστε να ενημερώνονται τα αποθηκευμένα δεδομένα εκτέλεσης, τα οποία χρησιμοποιούνται από τον χρονοδρομολογητή ώστε να προγραμματίζει τις μελλοντικές εκτελέσεις.

Συγκρίνοντας τις τιμές των καθυστερήσεων T1 και T2 για τα αντίστοιχα σενάρια αξιολόγησης, παρατηρούμε ότι η τιμή της καθυστέρησης T2 είναι αρκετά μεγαλύτερη από την T1, γεγονός αναμενόμενο, αν αναλογιστούμε τη φύση των εργασιών που περιλαμβάνονται στην δεύτερη περίπτωση καθώς αναφερόμαστε σε μεθόδους που αλληλεπιδρούν με μια σχεσιακή βάση και οι οποίες περιλαμβάνουν την εκτέλεση πολύπλοκων επερωτήσεων και τη συνεχή ενημέρωση δεδομένων.

Ακολουθούν δυο συγκριτικά διαγράμματα, σε καθένα από τα οποία παρουσιάζεται η διακύμανση των τιμών των καθυστερήσεων T1 και T2 αντίστοιχα για όλα τα σενάρια αξιολόγησης. Τα σενάρια απεικονίζονται ως ΣΑ1, ΣΑ2, ΣΑ3, ΣΑ4, ΣΑ5, ΣΑ6, ΣΑ7 και ΣΑ8 στον οριζόντιο άξονα.



Διάγραμμα 4: Οι τιμές της καθυστέρησης T1 για όλα τα σενάρια αξιολόγησης.



Διάγραμμα 5: Οι τιμές της καθυστέρησης T2 για όλα τα σενάρια αξιολόγησης.

Επίδραση της περιόδου στην απόδοση:

Πίνακας 17: Σενάριο Αξιολόγησης 2

Αναμενόμενη διάρκεια εκτέλεσης:		930000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**
714448,90 ms	120,63 ms	79443 ms

Πίνακας 18: Σενάριο Αξιολόγησης 3

Αναμενόμενη διάρκεια εκτέλεσης:		1740000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**
596888 ms	115,5 ms	84531 ms

Πίνακας 19: Σενάριο Αξιολόγησης 4

Αναμενόμενη διάρκεια εκτέλεσης:		1980000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**
712805,60 ms	115,8 ms	74213 ms

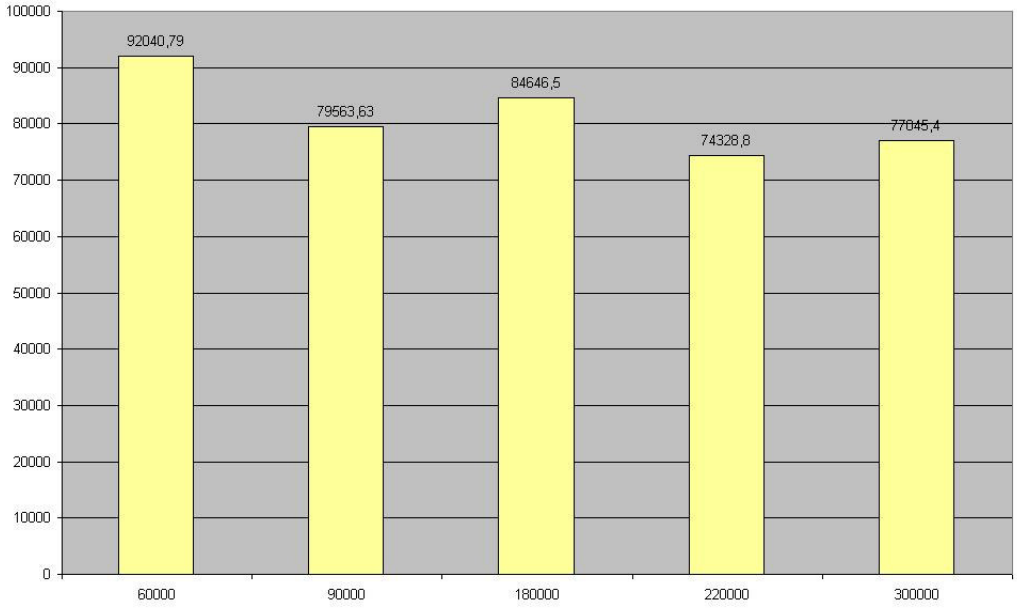
Πίνακας 20: Σενάριο Αξιολόγησης 5

Αναμενόμενη διάρκεια εκτέλεσης:		2700000 ms
Τυπική Απόκλιση Καθυστέρησης	**T1**	**T2**
965684,50 ms	117,4 ms	76928 ms

Στην περίπτωση του Quartz Enterprise Scheduler, τα συμπεράσματα που προκύπτουν από την ανάλυση των παραπάνω αποτελεσμάτων δεν οδηγούν σε ασφαλή συμπεράσματα ως προς το βαθμό επίδρασης της μεταβολής της περιόδου στην απόδοση του χρονοδρομολογητή κυρίως λόγω του ότι ο συγκεκριμένος

χρονοδρομολογητής παρουσιάζει μεγάλες καθυστερήσεις σε συνθήκες αυξημένου φόρτου εργασίας.

Το παρακάτω διάγραμμα περιγράφει τις διάφορες τιμές της ολικής καθυστέρησης στις διάφορες τιμές περιόδου μεταξύ των χρονοπρογραμματισμένων εργασιών, δηλαδή για τιμές περιόδου 60000, 90000, 180000, 220000 και 300000 ms.



Διάγραμμα 6: Η διακύμανση της ολικής καθυστέρησης για τις διάφορες τιμές περιόδου.

5.3 Σύγκριση επιδόσεων Polos Enterprise Scheduler και Quartz Enterprise Scheduler

Με βάση τα στοιχεία των πινάκων αποτελεσμάτων που παρατέθηκαν στην προηγούμενη ενότητα είμαστε πλέον σε θέση να προχωρήσουμε σε μια σύγκριση των δύο λύσεων χρονοπρογραμματισμού που μελετήθηκαν εκτενώς στην παρούσα εργασία.

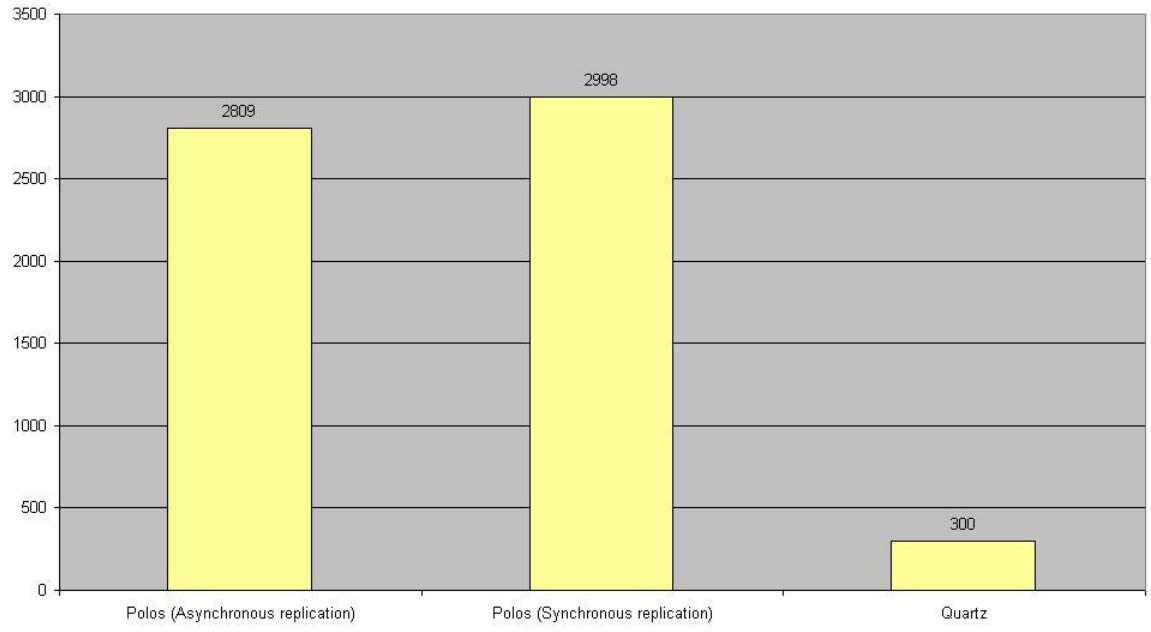
Ο βασικός άξονας της σύγκρισης αυτής είναι η μέγιστη ρυθμαπόδοση (throughput) που παρουσιάζουν οι χρονοδρομολογητές, δηλαδή ο μέγιστος αριθμός των εργασιών που μπορούν να εκτελεστούν σε διάστημα ενός λεπτού. Για τον καθορισμό των τιμών της ρυθμαπόδοσης, προσομοιώθηκαν «βαριά» σενάρια αξιολόγησης μέχρις ότου να

εντοπιστεί το όριο στο οποίο η τιμή της ρυθμαπόδοσης σταθεροποιείται αγγίζοντας τη μέγιστη δυνατή τιμή. Στην περίπτωση του Polos Enterprise Scheduler, η μέγιστη ρυθμαπόδοση μετρήθηκε για φόρτο 6000 ταυτόχρονων αιτήσεων, ενώ στην περίπτωση του Quartz Enterprise Scheduler, ο αντίστοιχος φόρτος είναι 4000 αιτήσεις.

Ακολουθεί πίνακας που παρουσιάζει τις τιμές της ρυθμαπόδοσης και των δυο χρονοδρομολογητών και το αντίστοιχο διάγραμμα των τιμών.

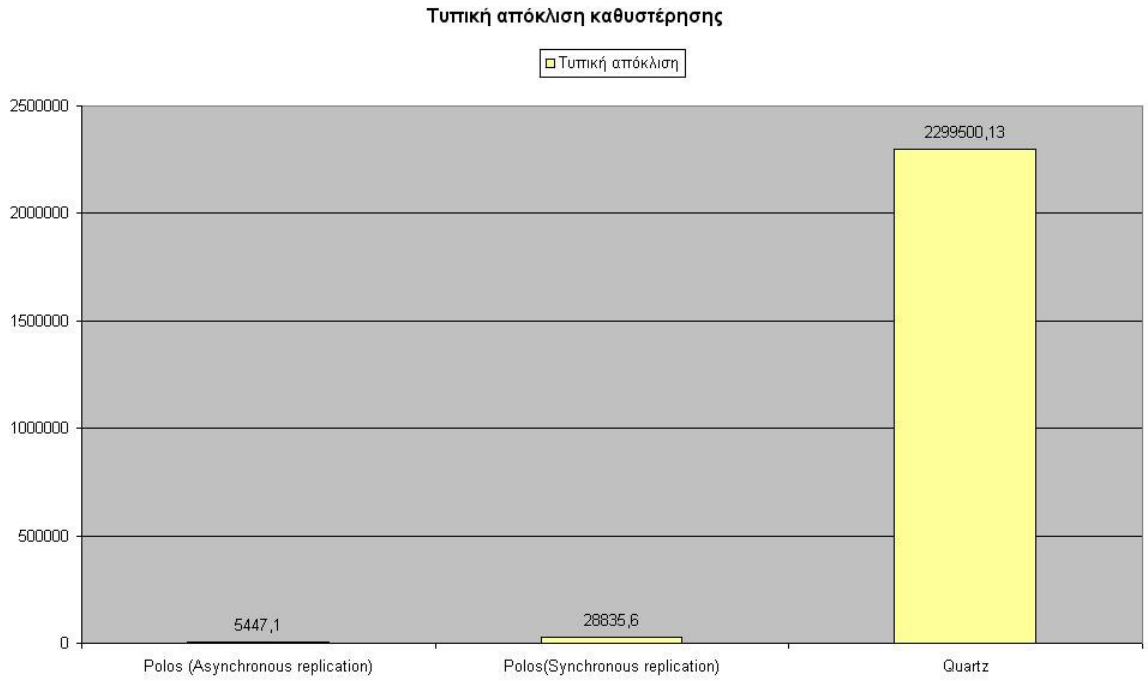
Πίνακας 21: Οι ρυθμαποδόσεις των δυο χρονοδρομολογητών.

Polos Enterprise Sheduler (Ασύγχρονη αντιγραφή)	2809
Polos Enterprise Sheduler (Σύγχρονη αντιγραφή)	2998
Quartz Enterprise Scheduler	300

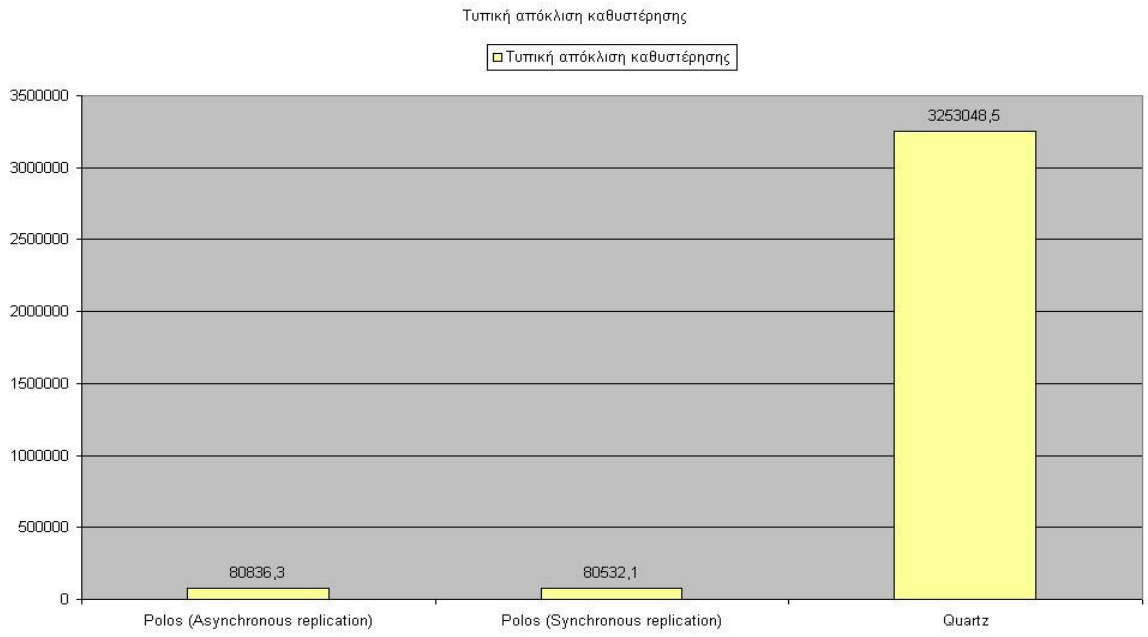


Διάγραμμα 7: Οι ρυθμαποδόσεις των δυο χρονοδρομολογητών.

Από το παραπάνω διάγραμμα εξάγεται το συμπέρασμα ότι στην περίπτωση του Polos Enterprise Scheduler και με τους δυο τύπους αντιγραφής η ρυθμαπόδοση είναι εντυπωσιακά μεγαλύτερη από εκείνη που παρουσιάζει ο Quartz Enterprise Scheduler. Γεγονός που δικαιολογείται από τα ακόλουθα διάγραμματα, στα οποία παρουσιάζονται οι τιμές των τυπικών αποκλίσεων των μέσων καθυστερήσεων, όπως προέκυψαν από τα αποτέλεσματα των μετρήσεων. Συγκεκριμένα, για τις ακόλουθες γραφικές απεικονίσεις, τα αποτελέσματα των σεναρίων των 3000 και 4000 παράλληλων αιτήσεων αξιοποιήθηκαν. Η επιλογή αυτή έγινε με κύριο γνόμωνα ότι αυτές οι συνθήκες μετρήσεων ήταν οι πιο ακραίες που προσομοιώθηκαν στα πλαίσια αυτής εργασίας. Ιδιαίτερο ενδιαφέρον παρουσιάζει η μεγάλη διαφορά των τιμών της τυπικής απόκλισης των δυο χρονοπρογραματιστών, η διαφορά αυτή δικαιολογεί και την διαφορά στην ρυθμαπόδοση των δυο υπό μελέτη συστημάτων. Η διαφορά αυτή έγκειται στην αρχιτεκτονική των δυο συστημάτων, και όπως έχει ήδη επισημανθεί, κυρίως στον ρόλο που διαδραματίζει η βάση δεδομένων στην λειτουργικότητα του Quartz Enterprise Scheduler, ο οποίος είναι η διαχείριση των χρονοπρογραμματιζόμενων εργασιών. Αντιθέτως, τον ρόλο αυτό, στην περίπτωση του Polos Scheduler, αναλαμβάνει ένας ο κατανεμημένος, μεταξύ των κόμβων της συστοιχίας, cache μηχανισμός, απαλάσοντας το σύστημα από το «βάρος» των συνεχών επερωτήσεων με τη βάση για τον συγχρονισμό των εργασιών σε κανονικές συνθήκες λειτουργίας αλλά και σε περίπτωση ανάκαμψης από σφάλμα του χρονοπρογραμματιστή (π.χ πιθανή διακοπή ρεύματος).



Διάγραμμα 8: Η τυπική απόκλιση της μέσης καθυστέρησης για το έβδομο σενάριο αξιολόγησης.



Διάγραμμα 9: Η τυπική απόκλιση της μέσης καθυστέρησης για το όγδοο σενάριο αξιολόγησης.

Η υψηλή αξιοπιστία των δύο χρονοπρογραμματιστών, η οποία αναλύεται στις συνιστώσες της ακρίβειας και της διαθεσιμότητας είναι γεγονός. Οι μετρήσεις μας έδειξαν ότι τα δύο συστήματα ανακάμπτουν από σημαντικά σφάλματα όπως η αποτυχία ενός κόμβου. Από τη μελέτη των αποτελεσμάτων των σεναρίων αξιολόγησης οδηγούμαστε στο συμπέρασμα ότι ο Polos Enterprise Scheduler πλεονεκτεί σε απόδοση σε σχέση με τον Quartz Enterprise Scheduler.

ΚΕΦΑΛΑΙΟ 6

ΕΠΙΛΟΓΟΣ

Στην παρούσα πτυχιακή εργασία, μελετήθηκαν δυο λύσεις χρονοπρογραμματισμού, ο Polos Enterprise Scheduler και ο Quartz Scheduler, που επιτρέπουν την χρονοπρογραμματισμένη εκτέλεση οποιασδήποτε εργασίας. Οι δυο αυτές λύσεις παρουσιάζουν το σύνολο εκείνων των χαρακτηριστικών που μπορούν να ανταποκριθούν ακόμα και στις πιο απαιτητικές προσδοκίες τόσο σε απόδοση όσο και σε αξιοπιστία. Πρόκειται για δυο scheduling αρχιτεκτονικές που σαφώς παρουσιάζουν ποικίλες διαφορές στην διάρθρωση τους αλλά και στον τρόπο επικοινωνίας των διάφορων components κατά τον χρονοπρογραμματισμό των εργασιών. Η σημαντικότερη από αυτές είναι ο ρόλος που παίζει η βάση δεδομένων στα δυο frameworks, καθώς στον Quartz Scheduler λειτουργεί ως η «καρδιά» της scheduling μηχανής ενώ στον Polos Enterprise Scheduler, το ρόλο αυτό φαίνεται να έχει η JBossCache, γεγονός πρωτότυπο για εφαρμογή που διαχειρίζεται σημαντικούς όγκους δεδομένων. Στο πρώτο τμήμα αυτής της μελέτης, αναλύονται αυτές τις δυο αρχιτεκτονικές, ενώ στο δεύτερο, επιχειρείται να πραγματοποιηθεί μια συγκριτική μελέτη των δυο συστημάτων όσων αφορά χαρακτηριστικά όπως η απόδοση, η διαθεσιμότητα, η αξιοπιστία και η κλιμάκωση. Όπως διαπιστώθηκε, κοινό χαρακτηριστικό των δυο λύσεων είναι η αξιόπιστη και η αποδοτική εκτέλεση των διαφόρων tasks, γεγονός που επιβεβαιώνει το μεγάλο εύρος των εφαρμογών που μπορούν να υποστηρίξουν, με τον Polos Enterprise Scheduler να αποδεικνύεται καταλληλότερος παρουσιάζοντας καλύτερη συμπεριφορά κατά τα διάφορα σενάρια αξιολόγησης που πραγματοποιήθηκαν.

ΠΑΡΑΡΤΗΜΑ Α΄

CONFIGURATIONS ΓΙΑ POLOS ENTERPRISE SCHEDULER

Προεγκατεστημένα:

Jboss 4.0.4 – http://www.jboss.org (Κατά προτήμιση το zip πακέτο)

Jboss Cache 1.2.4.SP2 – http://www.jboss.org/products/jbosscache

MySql 5.0 – http://www.mysql.com

Το πακέτο zip του Jboss περιέχει τρεις διαφορετικές εγκαταστάσεις του εξυπηρετητή (minimal, default και all). Επίσης, υπάρχει η δυνατότητα δημιουργίας νέας εγκατάστασης, προσαρμοσμένης στις απαιτήσεις του χρήστη.

Στην παρούσα μελέτη χρησιμοποιείται μια νέα εγκατάσταση του εξυπηρετητή με όνομα “myconfig”. Μια τέτοια εγκατάσταση δημιουργείται αντιγράφοντας την εγκατάσταση “all” από το μονοπάτι <JBOSS HOME>/server και επικολλώντας την στο ίδιο μονοπάτι, μετονομάζοντάς την σε “myconfig”. Η εκτέλεση αυτής της εγκατάστασης γίνεται δίνοντας την εντολή “run –c myconfig” στο μονοπάτι <JBOSS HOME>/bin από το command line.

DATASOURCE

Ο Polos Enterprise Scheduler απαιτεί τη δημιουργία μιας βάσης δεδομένων στον MySQL εξυπηρετητή, με όνομα “jbossdb”. Προκειμένου ο χρονοπρογραμματιστής να επικοινωνήσει με τη βάση αυτή απαιτείται η αντιγραφή του αρχείου mysql-ds.xml από το μονοπάτι <JBOSS_HOME>/docs/examples/jca στο <JBOSS_HOME>/server /myconfig/deploy . Ο κώδικας του αρχείου αυτού τροποποιείται έτσι ώστε το Jndi name να είναι “DefaultDS” και τα στοιχεία που αφορούν το χρήστη να ανταποκρίνονται σε αυτά της MySQL.

Ακολουθεί ενδεικτικό παράδειγμα του αρχείου:

```
<?xml version="1.0" encoding="UTF-8"?>

<datasources>
  <local-tx-datasource>
    <jndi-name>DefaultDS</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/jbossdb</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>USERNAME</user-name>
    <password>PASSWORD</password>
   <exception-sorter-class-
name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter</exception-
sorter-class-name>
    <!-- should only be used on drivers after 3.22.1 with "ping" support
    <valid-connection-checker-class-
name>org.jboss.resource.adapter.jdbc.vendor.MySQLValidConnectionChecker</valid
-connection-checker-class-name>
    -->
    <!-- sql to call when connection is created
    <new-connection-sql>some arbitrary sql</new-connection-sql>
      -->
    <!-- sql to call on an existing pooled connection when it is obtained from
pool - MySQLValidConnectionChecker is preferred for newer drivers
    <check-valid-connection-sql>some arbitrary sql</check-valid-connection-
sql>
      -->

    <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml
(optional) -->
    <metadata>
       <type-mapping>mySQL</type-mapping>
    </metadata>
  </local-tx-datasource>
</datasources>
```

Σχήμα 1: Παράδειγμα του αρχείου mysql-ds.xml.

Στη συνέχεια απαιτείται η τροποποίηση του “hsqldb-ds.xml”,που βρίσκεται στο μονοπάτι <JBOSS_HOME>/server/myconfig/deploy, έτσι ώστε το jndi name να είναι διαφορετικό από DefaultDS (π.χ. DefDS).

JMS

Επιπλέον, απαιτείται εγκατάσταση του JDBC driver της MySQl (http://dev.mysql.com/downloads/connector/j/5.0.html). Για να εγκατασταθεί ο JDBC driver αρκεί η αντιγραφή του αρχείου mysql-connector-java-5.0.4-bin.jar στο μονοπάτι <JBOSS_HOME>/server/myconfig/lib, καθώς και η αντιγραφή του αρχείου mysql-jdbc2-service.xml από το μονοπάτι <JBOSS_HOME>/docs/examples/jms στο <JBOSS_HOME>/server/myconfig/deploy-hashingleton/jms και η διαγραφή του αρχείου

από το πρώτο μονοπάτι, επιπλέον θα πρέπει το αρχείο να τροποποιηθεί κατάλληλα έτσι ώστε στο σημείο που έχει “MySQLDS” να υπάρχει “DefaultDS”.

Τελευταίο βήμα για μια επιτυχή εγκατάσταση είναι η τροποποίηση του αρχείου jbossmq-destinations-service.xml (βρίσκεται στο μονοπάτι <JBOSS_HOME>/server /myconfig/deploy-hashingleton/jms), έτσι ώστε να προστεθούν οι παρακάτω γραμμές, οι οποίες ορίζουν την ουρά μηνυμάτων μας.

```
<mbean code="org.jboss.mq.server.jmx.Queue" name="jboss.mq.destination:
service = Queue,name=PolosQ">
<depends optional-attribute-name="DestinationManager"> jboss.mq:service=
DestinationManager</depends>
<attribute name="ReceiversImpl">org.jboss.mq.server.
ReceiversImplArrayList
</attribute>
</mbean>
```

Σχήμα 2: Το jbossmq-destinations-service.xml αρχείο.

TREE CACHE

Για την εγκατάσταση της tree cache απαιτούνται τα παρακάτω βήματα:

- Αντιγραφή των βιβλιοθηκών jboss-cache.jar και jgroups.jar από τον φάκελο lib του JBossCache 1.2.4.SP2, στο μονοπάτι <JBOSS_HOME>/server/myconfig/lib.
- Αντιγραφή του αρχείου replSync-service.xml από το μονοπάτι JBossCache-1.2.4.SP2\etc\META-INF στο <JBOSS_HOME>/server/myconfig/deploy και τροποποίησή του έτσι ώστε η μεταβλητή loopback να έχει τιμή true για χρήστες Windows και αντικατάσταση του DummyTransactionManager με JBoss Transaction Manager.

Ακολουθεί παράδειγμα του αρχείου replSync-service.xml:

```
<attribute name="IsolationLevel">REPEATABLE_READ</attribute>

<attribute name="CacheMode">REPL_SYNC</attribute>

<attribute name="UseReplQueue">false</attribute>

<attribute name="ReplQueueInterval">0</attribute>

<attribute name="ReplQueueMaxElements">0</attribute>

<attribute name="ClusterName">TreeCache-Cluster</attribute>

 <attribute     name="ClusterConfig"><conf><UDP     mcast_addr="228.1.2.3"
mcast_port="48866"               ip_ttl="64"               ip_mcast="true"
mcast_send_buf_size="150000" mcast_recv_buf_size="80000"

 ucast_send_buf_size="150000"                  ucast_recv_buf_size="80000"
loopback="true"/>

 <PINGtimeout="2000"num_initial_members="3"up_thread="false"   down_thread
="false"/> <MERGE2 min_interval="10000" max_interval="20000"/> <FD_SOCK/>

<pbcast.STABLE desired_avg_gossip="20000"

 up_thread="false" down_thread="false"/>

<FRAG frag_size="8192" down_thread="false" up_thread="false"/>

<pbcast.GMS join_timeout="5000" join_retry_timeout="2000"

shun="true" print_local_addr="true"/>

<pbcast.STATE_TRANSFER up_thread="true" down_thread="true"/>

</config> </attribute>

<attribute name="FetchStateOnStartup">true</attribute>

<attribute name="InitialStateRetrievalTimeout">5000</attribute>

<attribute name="SyncReplTimeout">15000</attribute>

<attribute name="LockAcquisitionTimeout">10000</attribute>

<attribute name="EvictionPolicyClass"></attribute>

<attribute name="UseMarshalling">false</attribute>
```

Σχήμα 3: Το replSync-service.xml αρχείο.

POLOS ENTERPRISE SHEDULER

Η εγκατάσταση του Polos Scheduler στη συστοιχία γίνεται με την εισαγωγή του αρχείου PolosClusteredScheduler.ear σε κάποιο (τυχαίο) κόμβο. Με αυτόν τον τρόπο η εφαρμογή εγκαθίσταται σε κάθε κόμβο. Υπάρχει η δυνατότητα εισαγωγής χρονοδιαγραμμάτων μέσω του URL http://<JBoss_host>/Polos και με τις παραμέτρους:

- start (δέχεται ακέραιους αριθμούς σε ms από την τρέχουσα χρονική στιγμή)
- period
- reps
- rate
- usepdata
- jobclass
- jobtitle

Στα πλαίσια των ρυθμίσεων μας για την καταγραφή των επιμέρους ρυθμίσεων απαιτείται η δημιουργία ενός επιπλέον πίνακα, του 'polos_mem_table':

```
CREATE TABLE `polos_mem_table` (
  `pid` int(10) unsigned NOT NULL auto_increment,
  `sid` int(10) unsigned NOT NULL,
  `crep` int(10) unsigned NOT NULL,
  `firedAt` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `firedAtMs` bigint(20) unsigned NOT NULL,
  `realTime` bigint(20) unsigned NOT NULL,
  `mdbhash` varchar(100) NOT NULL,
  `triggerms` bigint(20) unsigned NOT NULL,
  `aftercachems` bigint(20) unsigned NOT NULL,
  PRIMARY KEY  (`pid`)) ENGINE=MEMORY DEFAULT CHARSET=greek
```

ΠΑΡΑΡΤΗΜΑ Β΄

CONFIGURATIONS ΓΙΑ QUARTZ SCHEDULER

Προεγκατεστημένα:

Jboss 4.0.4 – http://www.jboss.org

MySql 5.0 – http://ww.mysql.com

Quartz 1.6 - http://www.opensymphony.com/quartz/

Το πακέτο του Quartz περιλαμβάνει μια ομάδα jar αρχείων, που βρίσκονται μέσα στο φάκελο lib. Η βασική βιβλιοθήκη του Quartz ονομάζεται quartz.jar. Προκειμένου να χρησιμοποιηθεί σε μια εφαρμογή, αυτή η βιβλιοθήκη πρέπει να συμπεριληφθεί στο classpath, ή αν πρόκειται να αξιοποιηθεί στα πλαίσια ενός εξυπηρετητή, πρέπει να συμπεριληφθεί στο war ή ear αρχείο, αν πρόκειται για web ή enterprise εφαρμογή. Το επόμενο βήμα είναι ο καθορισμός των JobStores, τα οποία είναι υπεύθυνα για να κρατούν όλα εκείνα τα στοιχεία σχετικά με τις δουλειές, τα triggers, τα ημερολόγια και γενικά ότι πληροφορία δίνεις στον χρονοπρογραμματιστή. Υπάρχουν δυο ειδών JobStores:

- RAMJobStores, τα οποία κρατάνε όλα τα δεδομένα στην RAM, είναι ο πιο απλός τρόπος, με σαφώς καλύτερα αποτελέσματα σε απόδοση στη χρήση της CPU.
- JDBCJobStores, τα οποία κρατάνε όλα τα δεδομένα σε μια βάση μέσω JDBC.

Στην δική μας μελέτη, χρησιμοποιήθηκαν τα JDBCJobStores, καθώς μόνο μέσα από αυτά είναι δυνατή η λειτουργία του Quartz σε συστοιχία εξυπηρετητών. Ακολουθεί αναλυτική περιγραφή του τρόπου δημιουργίας της απαιτούμενης βάσης.

Datasource

Το πρώτο βήμα αποτελείται από τη δημιουργία μιας βάσης με το όνομα ΄myDS΄, στην οποία θα πρέπει να δημιουργηθούν οι εξής πίνακες:

- QRTZ_JOB_LISTENERS,
- QRTZ_TRIGGER_LISTENERS,
- QRTZ_LOCKS,
- QRTZ_FIRED_TRIGGERS,

- QRTZ_PAUSED_TRIGGER_GRPS,
- QRTZ_SCHEDULER_STATE,
- QRTZ_SIMPLE_TRIGGERS,
- QRTZ_CRON_TRIGGERS,
- QRTZ_BLOB_TRIGGERS,
- QRTZ_TRIGGERS,
- QRTZ_JOB_DETAILS,
- QRTZ_CALENDARS.

Ακολουθούν οι αντίστοιχες sql εντολές δημιουργίας των παραπάνω πινάκων:

```
DROP TABLE IF EXISTS QRTZ_JOB_LISTENERS;
DROP TABLE IF EXISTS QRTZ_TRIGGER_LISTENERS;
DROP TABLE IF EXISTS QRTZ_FIRED_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_PAUSED_TRIGGER_GRPS;
DROP TABLE IF EXISTS QRTZ_SCHEDULER_STATE;
DROP TABLE IF EXISTS QRTZ_LOCKS;
DROP TABLE IF EXISTS QRTZ_SIMPLE_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_CRON_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_BLOB_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_TRIGGERS;
DROP TABLE IF EXISTS QRTZ_JOB_DETAILS;
DROP TABLE IF EXISTS QRTZ_CALENDARS;

CREATE TABLE QRTZ_JOB_DETAILS
  (
    JOB_NAME  VARCHAR(80) NOT NULL,
    JOB_GROUP VARCHAR(80) NOT NULL,
    DESCRIPTION VARCHAR(120) NULL,
    JOB_CLASS_NAME   VARCHAR(128) NOT NULL,
    IS_DURABLE VARCHAR(1) NOT NULL,
    IS_VOLATILE VARCHAR(1) NOT NULL,
    IS_STATEFUL VARCHAR(1) NOT NULL,
    REQUESTS_RECOVERY VARCHAR(1) NOT NULL,
    JOB_DATA BLOB NULL,
    PRIMARY KEY (JOB_NAME,JOB_GROUP)
);

CREATE TABLE QRTZ_TRIGGERS
  (
    TRIGGER_NAME VARCHAR(80) NOT NULL,
    TRIGGER_GROUP VARCHAR(80) NOT NULL,
    JOB_NAME  VARCHAR(80) NOT NULL,
```

```
    JOB_GROUP VARCHAR(80) NOT NULL,
    IS_VOLATILE VARCHAR(1) NOT NULL,
    DESCRIPTION VARCHAR(120) NULL,
    NEXT_FIRE_TIME BIGINT(13) NULL,
    PREV_FIRE_TIME BIGINT(13) NULL,
    PRIORITY INTEGER NULL,
    TRIGGER_STATE VARCHAR(16) NOT NULL,
    TRIGGER_TYPE VARCHAR(8) NOT NULL,
    START_TIME BIGINT(13) NOT NULL,
    END_TIME BIGINT(13) NULL,
    CALENDAR_NAME VARCHAR(80) NULL,
    MISFIRE_INSTR SMALLINT(2) NULL,
    JOB_DATA BLOB NULL,
    PRIMARY KEY (TRIGGER_NAME,TRIGGER_GROUP),
    FOREIGN KEY (JOB_NAME,JOB_GROUP)
        REFERENCES QRTZ_JOB_DETAILS(JOB_NAME,JOB_GROUP)
);

CREATE TABLE QRTZ_JOB_LISTENERS
  (
    JOB_NAME  VARCHAR(80) NOT NULL,
    JOB_GROUP VARCHAR(80) NOT NULL,
    JOB_LISTENER VARCHAR(80) NOT NULL,
    PRIMARY KEY (JOB_NAME,JOB_GROUP,JOB_LISTENER),
    FOREIGN KEY (JOB_NAME,JOB_GROUP)
        REFERENCES QRTZ_JOB_DETAILS(JOB_NAME,JOB_GROUP)
);

CREATE TABLE QRTZ_SIMPLE_TRIGGERS
  (
    TRIGGER_NAME VARCHAR(80) NOT NULL,
    TRIGGER_GROUP VARCHAR(80) NOT NULL,
    REPEAT_COUNT BIGINT(7) NOT NULL,
    REPEAT_INTERVAL BIGINT(12) NOT NULL,
    TIMES_TRIGGERED BIGINT(7) NOT NULL,
    PRIMARY KEY (TRIGGER_NAME,TRIGGER_GROUP),
    FOREIGN KEY (TRIGGER_NAME,TRIGGER_GROUP)
        REFERENCES QRTZ_TRIGGERS(TRIGGER_NAME,TRIGGER_GROUP)
);

CREATE TABLE QRTZ_CRON_TRIGGERS
  (
    TRIGGER_NAME VARCHAR(80) NOT NULL,
    TRIGGER_GROUP VARCHAR(80) NOT NULL,
    CRON_EXPRESSION VARCHAR(80) NOT NULL,
    TIME_ZONE_ID VARCHAR(80),
    PRIMARY KEY (TRIGGER_NAME,TRIGGER_GROUP),
    FOREIGN KEY (TRIGGER_NAME,TRIGGER_GROUP)
        REFERENCES QRTZ_TRIGGERS(TRIGGER_NAME,TRIGGER_GROUP)
);
```

```
CREATE TABLE QRTZ_PAUSED_TRIGGER_GRPS
  (
    TRIGGER_GROUP  VARCHAR(80) NOT NULL,
    PRIMARY KEY (TRIGGER_GROUP)
);

CREATE TABLE QRTZ_BLOB_TRIGGERS
  (
    TRIGGER_NAME VARCHAR(80) NOT NULL,
    TRIGGER_GROUP VARCHAR(80) NOT NULL,
    BLOB_DATA BLOB NULL,
    PRIMARY KEY (TRIGGER_NAME,TRIGGER_GROUP),
    FOREIGN KEY (TRIGGER_NAME,TRIGGER_GROUP)
        REFERENCES QRTZ_TRIGGERS(TRIGGER_NAME,TRIGGER_GROUP)
);

CREATE TABLE QRTZ_TRIGGER_LISTENERS
  (
    TRIGGER_NAME  VARCHAR(80) NOT NULL,
    TRIGGER_GROUP VARCHAR(80) NOT NULL,
    TRIGGER_LISTENER VARCHAR(80) NOT NULL,
    PRIMARY KEY (TRIGGER_NAME,TRIGGER_GROUP,TRIGGER_LISTENER),
    FOREIGN KEY (TRIGGER_NAME,TRIGGER_GROUP)
        REFERENCES QRTZ_TRIGGERS(TRIGGER_NAME,TRIGGER_GROUP)
);

CREATE TABLE QRTZ_FIRED_TRIGGERS
  (
    ENTRY_ID VARCHAR(95) NOT NULL,
    TRIGGER_NAME VARCHAR(80) NOT NULL,
    TRIGGER_GROUP VARCHAR(80) NOT NULL,
    IS_VOLATILE VARCHAR(1) NOT NULL,
    INSTANCE_NAME VARCHAR(80) NOT NULL,
    FIRED_TIME BIGINT(13) NOT NULL,
    PRIORITY INTEGER NOT NULL,
    STATE VARCHAR(16) NOT NULL,
    JOB_NAME VARCHAR(80) NULL,
    JOB_GROUP VARCHAR(80) NULL,
    IS_STATEFUL VARCHAR(1) NULL,
    REQUESTS_RECOVERY VARCHAR(1) NULL,
    PRIMARY KEY (ENTRY_ID)
);

CREATE TABLE QRTZ_SCHEDULER_STATE
  (
    INSTANCE_NAME VARCHAR(80) NOT NULL,
    LAST_CHECKIN_TIME BIGINT(13) NOT NULL,
    CHECKIN_INTERVAL BIGINT(13) NOT NULL,
```

```
    PRIMARY KEY (INSTANCE_NAME)
);

CREATE TABLE QRTZ_LOCKS
  (
    LOCK_NAME  VARCHAR(40) NOT NULL,
    PRIMARY KEY (LOCK_NAME)
);


INSERT INTO QRTZ_LOCKS values('TRIGGER_ACCESS');
INSERT INTO QRTZ_LOCKS values('JOB_ACCESS');
INSERT INTO QRTZ_LOCKS values('CALENDAR_ACCESS');
INSERT INTO QRTZ_LOCKS values('STATE_ACCESS');
INSERT INTO QRTZ_LOCKS values('MISFIRE_ACCESS');
```

Επιπλέον, για την αποθήκευση των διαφόρων μετρήσεων, με βάση τις οποίες πραγματοποιήθηκε η αξιολόγηση των δυο συστημάτων, δημιουργήθηκε ο πίνακας ‘quartz_mem_table’:

```
CREATE TABLE `quartz_mem_table` (

  `pid` int(10) unsigned auto_increment,

  `beforeFired` bigint(20) unsigned,

  `afterFired` bigint(20) unsigned,

  `executionTime` bigint(20) unsigned,

  `realTime` varchar(100),

   `realTimeAtms` varchar(100),

   `triggerNm` varchar(100),

   `JobNm` varchar(100),

   PRIMARY KEY  (`pid`)

)  DEFAULT CHARSET=greek
```

Clustering

Προκειμένου να υπάρχει η δυνατότητα να λειτουργήσει ο Quartz σε μια συστοιχία, πρέπει απλά να τροποποιηθούν κατάλληλα κάποιες properties στο quartz.properties αρχείο. Συγκεκριμένα, πρέπει να συμπεριληφθούν οι εξής properties στο προαναφερθέντα αρχείο:

- org.quartz.scheduler.instanceId = auto
- org.quartz.jobStore.isClustered = true

Κάθε χρονοπρογραμματιστής που θα συμμετέχει στη συστοιχία πρέπει να έχει ένα αντίγραφο του quartz.properties, το οποίο είναι ίδιο σε όλους, με εξαίρεση την τιμή που καθορίζει το μέγιστο αριθμό των νημάτων, το οποίο καθορίζεται από τη τιμή του org.quartz.threadPool.threadCount.

Quartz Properties

Επόμενο βήμα είναι η τροποποίηση του αρχείου quartz.properties που βρίσκεται στο quartz-all-1.6.0.jar, ώστε να ρυθμιστούν τα εξής:

- Η επικοινωνία του Quartz Scheduler με την βάση που δημιουργήθηκε στο παραπάνω βήμα.
- Το clustered περιβάλλον στο οποίο λειτουργεί ο Quartz Scheduler.

Ακολουθεί το αρχείο που χρησιμοποιήθηκε στην παρούσα μελέτη:

```
org.quartz.scheduler.instanceName = DefaultQuartzScheduler

org.quartz.scheduler.instanceId = AUTO
org.quartz.scheduler.rmi.export = false
org.quartz.scheduler.rmi.proxy = false
org.quartz.scheduler.wrapJobExecutionInUserTransaction = false
org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
org.quartz.threadPool.threadCount = 500
org.quartz.threadPool.threadPriority = 5
org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializin
gThread = true
org.quartz.jobStore.misfireThreshold = 60000
org.quartz.jobStore.class =
org.quartz.impl.jdbcjobstore.JobStoreCMT
org.quartz.jobStore.driverDelegateClass =
org.quartz.impl.jdbcjobstore.StdJDBCDelegate
org.quartz.jobStore.tablePrefix = QRTZ_
org.quartz.jobStore.dataSource = myDS
org.quartz.dataSource.myDS.driver = com.mysql.jdbc.Driver
org.quartz.dataSource.myDS.URL = jdbc:mysql://localhost:3306/myDS
org.quartz.dataSource.myDS.user = root
org.quartz.dataSource.myDS.password = 12345
org.quartz.dataSource.myDS.maxConnections = 5
org.quartz.jobStore.isClustered = true
```

Σχήμα 1: Παράδειγμα του αρχείου quartz.properties.

ΟΡΟΛΟΓΙΑ

Throughput	Ρυθμαπόδοση
Standard Deviation	Τυπική απόκλιση
Computer Cluster	Συστοιχία Υπολογιστών
Instance	Υπόσταση ενός αντικειμένου
Batch Jobs	Ομάδα εργασιών
Fail-Over	Ανάκαμψη από αποτυχία
Fault-Tolerance	Ανοχή στα σφάλματα χρόνου εκτέλεσης
High Availabillity	Υψηλή Διαθεσιμότητα
Interface	Διεπαφή
Load Balancing	Κατανομή φόρτου
Scalability	Κλιμάκωση
Multithreading	Πολυνηματικός
Persistent Storage	Μόνιμη Αποθήκευση
Replication	Πανομοιότυπη αντιγραφή – Επανάληψη μίας δράσης
Node	Κόμβος
Workers Threads	Νήματα εργάτες
Task	Ατομικό τμήμα ή βήμα μίας εργασίας

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΤΙΚΟΛΕΞΑ

API	Application Programming Interface
RMI	Remote Method Invocation
JTA	Java Transaction API
JMX	Java Management Extensions
JMS	Java Messaging System
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
JVM	Java Virtual Machine
EJB	Enterprise Java Beans
MDB	Message Driven Bean

ΑΝΑΦΟΡΕΣ

1. Vassileios Tsetsos, Odysseas Sekkas, Ioannis Priggouris, Stathes Hadjiefthymiades, 2006, "A scheduling framework for enterprise services", Journal of Systems and Software archive Vol 79 , pp: 259 - 272
2. Ευστράτιος Παυλάκης, Μελέτη Τεχνικών Χρονοδρομολόγησης σε Συστοιχίες Εξυπηρετών Εφαρμογών, διπλωματική εργασία *στο μεταπτυχιακό πρόγραμμα* σπουδών του τμήματος Πληροφορικής και Τηλεπικοινωνιών, ΕΚΠΑ
3. R Cattell 2000. Java 2 Platform, Enterprise Edition: platform and component specifications. Addison-Wesley Pub Co, USA.
4. G Dale, 2005. Kronova eScheduler Concepts Guide. Indus Consultancy Services,
5. Ivelin Ivanov, O'Reilly On Java Com 2003, J2EE Clustering with JBoss
6. Dejan Bosanac, O'Reilly On Java Com 2004, Job Scheduling in Java
7. D. Feitelson, 1998. Metrics and benchmarking for parallel job scheduling. In: Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science, Volume 1459. Springer-Verlag, Pages 1–24.
8. Java Transaction API (JTA) Specification 1.0.1, Sun Microsystems, Available At : http://java.sun.com/products/jta/
9. J2EE Connector Architecture (JCA), Sun Microsystems, Available At: http://java.sun.com/j2ee/connector/
10. David Sims, 2005, Flux Workflow Engine, Job Scheduler, Available At: http://freshmeat.net/projects/flux-workflow/
11. Timer for Application Servers Java Specification Request, Java Community Process 2003, Available At: http://jcp.org/en/jsr/detail?id=236
12. J. Goodwill, 2001. Apache Jakarta-Tomcat. Apress Publishing, Berkeley.
13. C.Gu, 2003. The complete log4j manual. QOS.ch, Lausanne, Switzerland.

for Experimental Design, Measurement, Simulation, and Modeling. Wiley-

Interscience, New York, USA.

14. Joy, 2000. Java(TM) Language Specification, second ed. Addison-Wesley Pub Co
15. Roman, 2002. Mastering Enterprise JavaBeans, third ed. Wiley Computer Publishing, USA. Sims, D., 2002. Job Scheduling in J2EE Applications. Sims Computing, Inc., USA.
16. Stark, 2002. Jboss Administration and Development, second ed. Jboss Group LLC, Atlanta, USA. Tyagi, S. et al., 2003. Core Java Data Objects.
17. Bill Burke, Sacha Labourey, O' Reilly On Java Com 2002, Clustering with JBoss 3.0
18. Jboss 4 guide, JBoss 2005, Available At: http://docs.jboss.org/j
19. Frank Sommers, The Server side com 2003, JBoss High-availability and Clustering Article, Available At: http://www.theserverside.com/news/
20. JBoss Clustering Patterns Library, JBoss Wiki 2006, Available At: http://wiki.jboss.org/wiki/
21. Java Management Extensions (JMX) Remote API, Java Specification 2003-2006, Available At: http://jcp.org/aboutJava/communityprocess/final/jsr160/index.html
22. Jboss Cache 1.4.0 Jalapeno Tutorial and User Documentation , JBoss ORG, Available At: http://labs.jboss.com/portal/jbosscache/docs/index.html
23. Crontab Tutorial, Available At: http://en.wikipedia.org/wiki/Crontab
24. Anacron -- anac(h)ronistic cron, Available At: http://anacron.sourceforge.net/
25. The Java Language Environment Whitepaper, Available At: http://java.sun.com/docs/white/index.html
26. Jcrontab- a Java Cron Scheduler, Available At: http://www.jcrontab.org/
27. JBoss Application Server, Available At: http://labs.jboss.com/portal/jbossas/?prjlist=false
28. Bill Burke, Adrian Brock O' Reilly On Java Com 2003, Aspect-Oriented Programming and JBoss, Available At:

http://www.onjava.com/pub/a/onjava/2003/05/28/aop_jboss.html

29. MySQL 5.0 RDBMS , Available At: http://www.mysql.com

30. NetBeans 5.0 Java IDE, Available At: www.netbeans.org

31. Java Messaging Service (JMS), Available At: http://java.sun.com/products/jms/

32. Quartz Framework, Available At:

http://www.onjava.com/pub/a/onjava/2005/09/28/what-is-quartz.html

33. Unix Man, Available At: http://www.rt.com/man/

34. Crontab Tutorial, Available At: http://en.wikipedia.org/wiki/Crontab

35. Bhattacharya, B., 2003. “Pulsar Scheduler J2EE”