Ατομική Διπλωματική Εργασία

A FRAMEWORK FOR MONITORING SERVER BEHAVIOUR AND ITS APPLICATIONS

Χρίστος Σέας

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2019

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

A FRAMEWORK FOR MONITORING SERVER BEHAVIOUR AND ITS APPLICATIONS

Χρίστος Σέας

Επιβλέπων Καθηγητής

Γιαννάκης Σαζεΐδης

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

Μάιος 2019

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να αφιερώσω μία σελίδα σε όλους αυτούς που με βοήθησαν και με στήριξαν στην δημιουργία και ολοκλήρωση της ατομικής διπλωματικής εργασίας μου. Η

βοήθεια και η στήριξη αυτών των ανθρώπων ήταν ιδιαίτερα σημαντική για εμένα και δεν θα μπορούσα να τους παραλείψω.

Πρωτίστως, θα ήθελα να ευχαριστήσω ιδιαιτέρως τον επιβλέποντα καθηγητή μου, Δρ. Γιαννάκη Σαζεΐδη, για την ευκαιρία που μου έδωσε να εκπονήσω μαζί του την ατομική διπλωματική μου εργασία. Τον ευχαριστώ που με ένταξε στην ομάδα του UniServer και για την εμπιστοσύνη που έδειξε στο πρόσωπο μου, ακόμα κι στις πιο δύσκολες στιγμές. Ακόμη, η καθοδήγηση του κατά την διάρκεια της διπλωματικής μου εργασίας ,με τον δικό του τρόπο και με το παράδειγμά του, ήταν καθοριστική για την πρόοδο μου σαν απόφοιτος Πληροφορικής. Τον ευχαριστώ επίσης για τις συμβουλές του, τόσο σε ακαδημαϊκό επίπεδο όσο και σε προσωπικό και την αστείρευτη υπομονή και καλή διάθεσή του. Τέλος, το φιλικό περιβάλλον των συναντήσεων μας και του εργαστηρίου με έκανε να νιώθω άνετα και να μπορώ να εκφραστώ ελεύθερα.

Θα ήθελα επίσης να ευχαριστήσω τους διδακτορικούς φοιτητές Ζαχαρία Χατζηλάμπρου και Παναγιώτα Νικολάου που συνεχώς μου έδιναν την στήριξη, την καθημερινή βοήθεια και τις συμβουλές και οδηγίες τους. Η βοήθεια τους ήταν σημαντική κατά την διάρκεια των δυσκολιών που αντιμετώπιζα στην ατομική διπλωματική μου εργασία. Τέλος, θα ήθελα να ευχαριστήσω τους συμφοιτητές μου Αντρέα Πρόξενο, Μάριο Κελεπέσιη και Γεωργία Αντωνίου για την στήριξη τους κατά την διάρκεια της διπλωματικής μου εργασίας και το φιλικό κλίμα που προσέδιδαν στο εργαστήριο.

Σας ευχαριστώ όλους.

Περίληψη

Η ανάπτυξη και βελτίωση των υπολογιστών είναι η αιτία της ραγδαίας τεχνολογικής ανάπτυξης της κοινωνίας μας. Οι υπολογιστές ολοένα και βελτιώνονται και οι ερευνητές χρειάζονται πρακτικά εργαλεία ώστε να έχουν την δυνατότητα να αντιλαμβάνονται καλύτερα την συμπεριφορά του υπολογιστή σε συνθήκες πειραμάτων. Με την χρήση αυτών των εργαλείων και την καλύτερη αντίληψη των υπολογιστών, δίνεται η δυνατότητα στους ερευνητές για έρευνα στην εξοικονόμηση ενέργειας, καλύτερης επίδοσης, ανεύρεση λόγων crash του συστήματος κ.τ.λ.

Στην παρούσα ατομική Διπλωματική Εργασία θα υλοποιήσουμε ένα framework για παρακολούθηση της συμπεριφοράς του υπολογιστή σε συνθήκες πειραμάτων. Συγκεκριμένα θα αναλύσουμε τι χρειαζόμαστε για να παρακολουθήσουμε την συμπεριφορά του υπολογιστή, τον τρόπο με τον οποίο μπορούμε να τον παρακολουθήσουμε και την σωστή διαδικασία ώστε οι παρατηρήσεις μας να είναι σωστά δομημένες και καταγεγραμμένες.

Στην συνέχεια θα αναφερθούμε στην χρήση του εργαλείου. Χρησιμοποιήσαμε το εργαλείο στα πλαίσια ενός διαπανεπιστημιακού προγράμματος (Uniserver) για να χαρακτηρίσουμε (characterization) μία εφαρμογή και να κάνουμε μία ανάλυση δεδομένων (Data analysis) για ανεύρεση λόγων crash του συστήματος. Θα αναδείξουμε πως το εργαλείο μας έχει δώσει επιπλέον δυνατότητες στο characterization της εφαρμογής και στην ανάλυση των δεδομένων μας.

Τέλος θα εξάγουμε τα συμπεράσματα τα οποία προκύπτουν από όλη την έκταση της Διπλωματικής Εργασίας και θα προτείνουμε την μελλοντική εργασία που πρέπει να επιτελεσθεί.

Περιεχόμενα

Κεφάλαιο 1

1. Εισαγωγή

1.1 Γενική Εισαγωγή

Υπολογιστές δεν βρίσκονται πλέον μόνο στις μεγάλες επιχειρήσεις και σαν προσωπικοί υπολογιστές στα σπίτια μας. Είναι παντού στην καθημερινότητά μας· στις τηλεοράσεις μας, στα κινητά μας τηλέφωνα, στις πλείστες μοντέρνες ηλεκτρικές συσκευές κ.α.

Η σχεδίαση σύγχρονων υπολογιστών και η αποδοτική αξιοποίηση τους απαιτεί την συνεχή βελτίωσή τους σε όλους τους τομείς. Την συνεχή αύξηση της επίδοσης και της απόδοσης, την μείωση της κατανάλωσης ενέργειας καθώς και την προσθήκη νέων μηχανισμών. Αυτό απαιτεί από τους σχεδιαστές την συνεχή έρευνα για την ανάπτυξη νέων μηχανισμών. Για να μπορέσουν να πραγματοποιηθούν αυτές οι έρευνες ώστε να βελτιωθούν οι υπολογιστές, πρέπει να παρέχονται εργαλεία στους σχεδιαστές που θα τους δίνουν την δυνατότητα να παρατηρούν λεπτομερώς την συμπεριφορά του υπολογιστή. Αυτά τα εργαλεία είναι απαραίτητα ώστε οι σχεδιαστές παρατηρώντας την συμπεριφορά του υπολογιστή, να κατανοήσουν καλύτερα την συμπεριφορά του, να επιβεβαιώσουν ή να απορρίψουν θεωρίες ερευνών τους και να εξαγάγουν καινούργια συμπεράσματα για λειτουργίες του υπολογιστή. Επομένως αυτά τα εργαλεία είναι πολύ σημαντικά ώστε να συνεχίσει η βελτίωση των υπολογιστών μας σε όλους τους τομείς.

1.2 Στόχος Διπλωματικής Εργασίας

Ο κύριος στόχος της παρούσας Διπλωματικής Εργασίας είναι η υλοποίηση ενός framework παρατήρησης της συμπεριφοράς του υπολογιστή. Οι χρήστες του εργαλείου θα είναι έμπειροι χρήστες υπολογιστών, κυρίως ερευνητές και σχεδιαστές υπολογιστών. Επιθυμούμε το εργαλείο αυτό να:

- Παρέχει μετρήσεις διαφόρων μετρικών στον χρήστη σχετικά με την συμπεριφορά του υπολογιστή.
- Παρέχει εύκολο τρόπο καταχώρησης πειραμάτων (benchmarks).
- Παρέχει χρονική στιγμή για κάθε μέτρηση που καταχωρήθηκε ώστε να γνωρίζουμε ανά πάσα στιγμή τι έτρεχε στον υπολογιστή και ποιες μετρήσεις πήραμε.

Έχουμε ήδη χρησιμοποιήσει το εργαλείο αυτό στο διαπανεπιστημιακό πρόγραμμα UniServer. Το UniServer Project έχει σκοπό να διευκολύνει την εμφάνιση των λύσεων IoT και Smart City μέσω της υιοθέτησης μιας κατανεμημένης υποδομής, όπου οι αποφάσεις λαμβάνονται πιο τοπικά με την προηγμένη Micro-Server Technology. Χρησιμοποιήσαμε το εργαλείο στο Uniserver Project για τον καλύτερο χαρακτηρισμό (characterization) της Polaris εφαρμογής του Project. Το UniServer Project θα χρησιμοποιήσει την εισήγηση μας για το πώς πρέπει να τρέχουμε την εφαρμογή, την οποία παράξαμε μέσω του characterization μας, χρησιμοποιώντας το framework. Επιπλέον χρησιμοποιήσαμε το framework για μία αρχική ανάλυση δεδομένων από τις μετρήσεις που παίρνουμε, ώστε αν μπορούμε να εξάγουμε κάποιο συμπέρασμα για πιθανά crashes σε ένα σύστημα.

Κεφάλαιο 2

2. Αρχιτεκτονική Υλικού (Hardware Architecture)

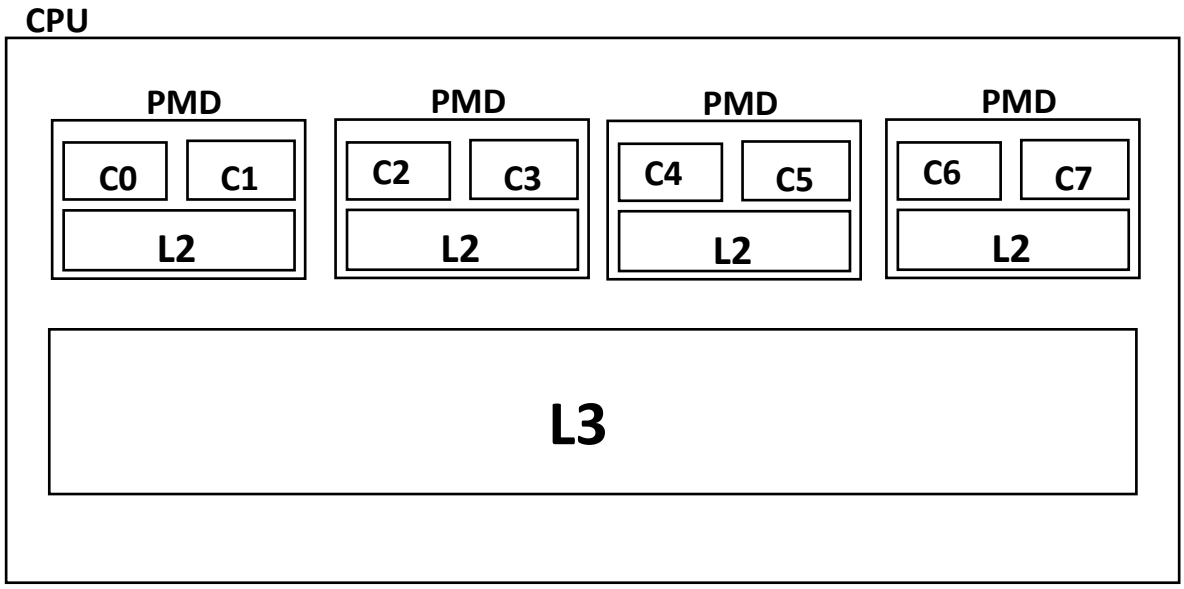
2.1 Γενικά

Ολόκληρη η μελέτης δημιουργήθηκε σε πειράματα και μετρήσεις που έγιναν σε δύο servers στο εργαστήριο· τον X-Gene 2 και τον X-Gene 3 της AppliedMicro. Και στους δύο servers έτρεξαν σωρεία πειραμάτων με την μεθοδολογία που αναπτύχθηκε και πάρθηκαν μετρήσεις. Λόγω απαιτήσεων του UniServer Project, το characterization της Polaris εφαρμογής έγινε στον X-Gene 2 server.

2.2 X-Gene 2 Server

Η X-Gene είναι μια οικογένεια μικροεπεξεργαστών ARM υψηλής απόδοσης από την AppliedMicro.Η πνευματική ιδιοκτησία X-Gene , πωλήθηκε τελικά στην Ampere Computing, η οποία το ανασχεδίασε και το βελτίωσε κάτω από την οικογένεια eMAG.

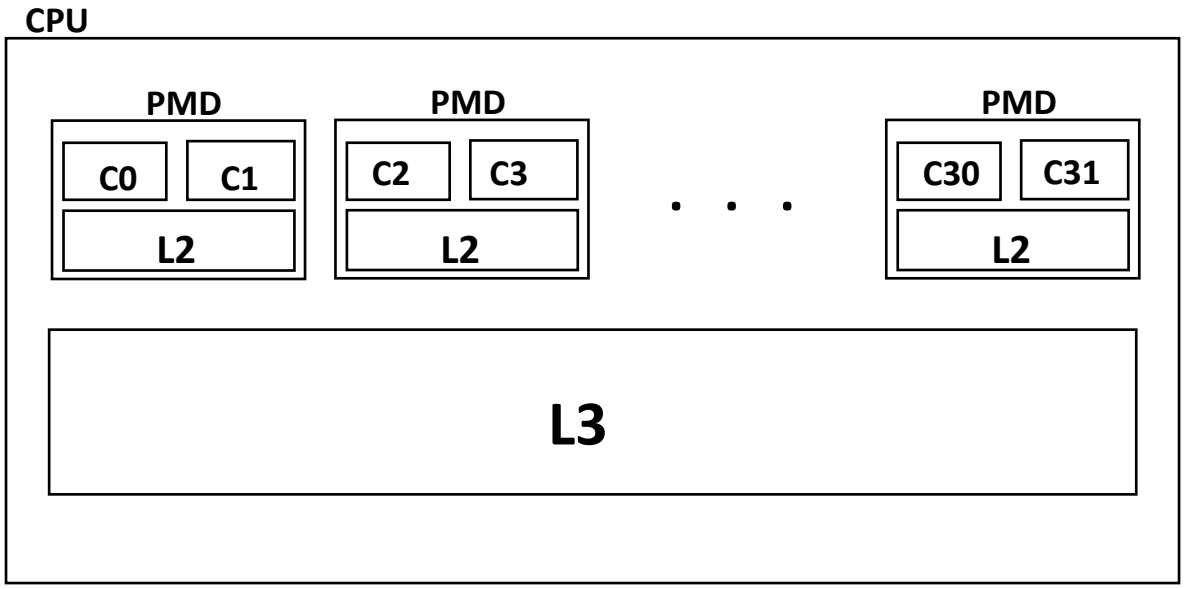
Οι X-Gene 2 servers ανακοινώθηκαν τον Σεπτέμβριο του 2014 και λανσαρίστηκαν στις αρχές του 2015. Οι επεξεργαστές X-Gene 2 βασίζονται στη μικροαρχιτεκτονική Shadowcat. Έχουν 8 φυσικούς πυρήνες που λειτουργούν μέχρι και σε 2.4 GHz, 4 memory channels και 3 επίπεδα Cache (L1, L2, L3 cache). Ο κάθε ένας από τους πυρήνες έχει δική του L1 Data Cache και L1 Instruction Cache. Οι 8 φυσικοί πυρήνες τοπολογούνται σε 4 PMDs – 2 πυρήνες για κάθε PMD. Κάθε ένα PMD έχει 2 φυσικούς πυρήνες και μία L2 Cache την οποία μοιράζονται οι 2 πυρήνες που ανήκουν στο συγκεκριμένο PMD. Επίσης, όλοι οι πυρήνες έχουν και μια κοινή L3 Cache 8MiB την οποία μοιράζονται. Πιο κάτω φαίνεται διαγραμματικά η αρχιτεκτονική του X-Gene 2 processor.



X-Gene 3 Server

Οι X-Gene 3 server ανακοινώθηκαν το 2016 και λανσαρίστηκαν το 2017. Οι επεξεργαστές X-Gene 3 βασίζονται στη μικροαρχιτεκτονική Skylark. Έχουν 32 φυσικούς πυρήνες που λειτουργούν μέχρι και σε 3 GHz, 8 memory channels και 3 επίπεδα Cache (L1, L2, L3 cache). Όπως και στον X-Gene 2, ο κάθε ένας από τους πυρήνες έχει δική του L1 Data Cache και L1 Instruction Cache και οι 32 πυρήνες τοπολογούνται σε 16 PMDs – 2 πυρήνες για κάθε PMD (κάθε PMD έχει 2 φυσικούς πυρήνες και μια L2 Cache που μοιράζονται όπως τον X-Gene 2). Όλοι οι πυρήνες έχουν και μια κοινή L3 Cache την

οποία μοιράζονται. Πιο κάτω φαίνεται διαγραμματικά η αρχιτεκτονική του X-Gene 3 processor.



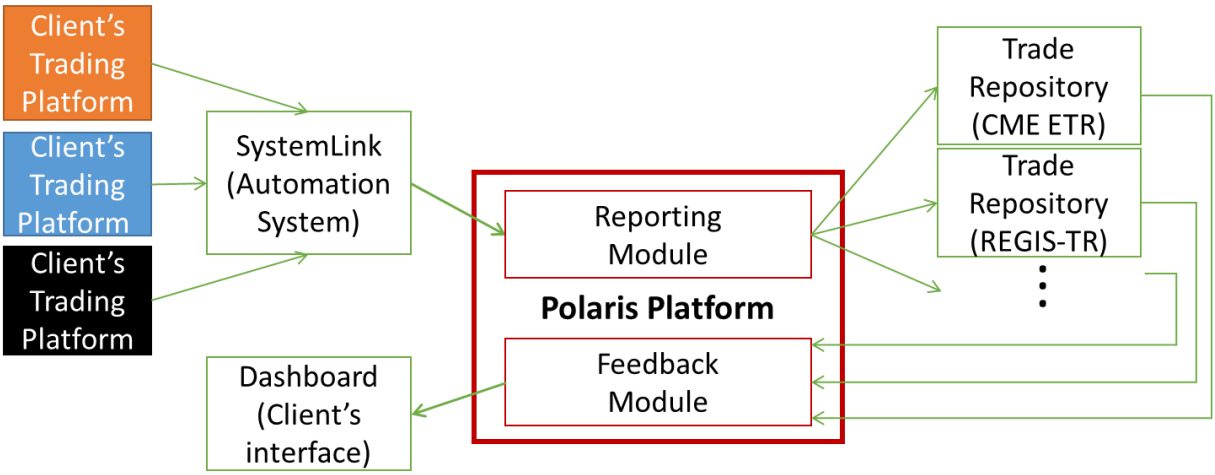
Κεφάλαιο 3

3. Polaris Application

3.1 Γενικά

Το Polaris Application είναι μία εφαρμογή της εταιρίας **Meritorious Audit Ltd** που στεγάζεται στην Κύπρο. Η εταιρία συμμετέχει στο Ευρωπαϊκό Project UniServer. Έχει υλοποιήσει την εφαρμογή Polaris που είναι ένα Regulatory reporting application. Το application δημιουργήθηκε βάσει κανονισμού για την υποδομή των ευρωπαϊκών αγορών (European Markets Infrastructure Regulation -EMIR). Ο κανονισμός τέθηκε σε ισχύ στις 16 Αυγούστου 2012 και εισήγαγε απαιτήσεις που αποσκοπούν στη βελτίωση της διαφάνειας των Over-The-Counter (OTC) derivatives και στη μείωση των κινδύνων που συνδέονται με αυτές τις αγορές. Όλες οι συναλλαγές πρέπει πλέον, να κάνουν αναφορά σε Trading Repositories (TRs).

Η εφαρμογή ουσιαστικά λειτουργά προσπαθώντας να δώσει διαφάνεια των Over-The-Counter (OTC) derivatives καταγράφοντας τις συναλλαγές. Αυτό απαιτεί την επεξεργασία και επικύρωση μεγάλης ποσότητας πληροφορίας και τον χειρισμό ευαίσθητων προσωπικών δεδομένων των πελατών. Πιο κάτω φαίνεται διαγραμματικά, πως γίνεται η επικύρωση μιας συναλλαγής



3.2 Λειτουργεία Polaris

Η Polaris εφαρμογή είναι γραμμένη στην γλώσσα προγραμματισμού Perl. Η εφαρμογή αποτελείται από τρις φάσεις

1. Initialization
2. Validation
3. Feedback

Για σωστή λειτουργία του application χρειάζονται και οι τρεις φάσεις που τρέχουν με την σειρά. Παρόλα αυτά, η ποιότητα εξυπηρέτησης (Quality of Service - QoS) της εφαρμογής εξαρτάται από την Validate φάση. Η εφαρμογή ανάλογα με το configuration που θα της

δώσουμε τρέχει με διαφορετικά data sets:small, medium, large. Στο τέλος δίνει μία αναφορά, για το πόσα transactions εκτέλεσε στην validate φάση (αυτό είναι σταθερό, αναλόγως του data set) και σε πόσο χρόνο, οπότε έτσι μπορούμε να γνωρίζουμε τα transactions/min.

3.3 Quality of Service

Στην δικιά μας περίπτωση μας απασχολεί μόνο το QoS για το small data set. Το small data set εκτελεί 2591 transactions. Σύμφωνα με το QoS, πρέπει να εκτελούμε **1800 transactions/min**· επομένως το small data set χρειάζεται να τρέχει σε λιγότερο από **86.37sec**. Στον πιο κάτω πίνακα μπορούμε να δούμε τα μετρικά που πρέπει να τηρούμε στο ελάχιστο αν θέλουμε να είμαστε στα όρια του QoS.

Metric	**Description**
Throughput	1800 transactions/min per instance **(86.37 sec)**
Availability	99%
Accuracy	100%

3.4 UniServer Project

3.4.1 Γενικά

Το UniServer έχει το όνομα του από της λέξεις **Uni**versal και **Server.** Το UniServer Project είναι επιχορηγημένο από την Ευρωπαϊκή Ένωση και έχει ξεκινήσει Φεβρουάριο 2016. Το UniServer είναι ένα τριετές πρόγραμμα που απονέμεται από το πρόγραμμα Έρευνα και Καινοτομία Horizon 2020. Αναμένεται να ολοκληρωθεί Ιούλιο 2019.

Το Uniserver επιδιώκει την ανάπτυξη:

- Ενός ευρέως διαδεδομένου συστήματος αρχιτεκτονικής υπολογιστών
- Ενός οικοσυστήματος λογισμικού για servers

Το UniServer σκοπεύει στο να αλλάξει την εξέλιξη του Διαδικτύου από μία υποδομή όπου τα δεδομένα συγκεντρώνονται σε data centers, σε μία υποδομή όπου η επεξεργασία δεδομένων γίνεται με ένα κατανεμημένο και τοπικό τρόπο κοντά στις πηγές δεδομένων. Σκοπεύει να υλοποιήσει τον κύριο του στόχο βελτιώνοντας σημαντικά την ενεργειακή αποδοτικότητα, την επίδοση, την αξιοπιστία και την ασφάλεια των σύγχρονων servers, ενώ ταυτόχρονα θα ενισχύσει το λογισμικό του συστήματος.

Το UniServer σκοπεύει στο να αναπτύξει αποτελεσματικούς τρόπους για να αποκαλύψει την ετερογένεια του υλικού που προκαλείται από τις διαφορές μεταξύ των παραγόμενων επεξεργαστών. Έπειτα, χρησιμοποιώντας αυτούς τους τρόπους, να τους αξιοποιήσει για την βελτίωση της ενεργειακής απόδοσης και της επίδοσης του συστήματος.

Για να επιτευχθεί αυτό, θα ενσωματωθεί λογισμικό στο σύστημα. Το λογισμικό θα είναι ελαφρύ (lightweight) για να μην έχει επιπτώσεις στην επίδοση του server. Το λογισμικό που θα ενσωματωθεί, θα θέτει στο σύστημα όρια τάσης και συχνότητας για κάθε εφαρμογή που θα τρέχει το σύστημα. Αυτά τα όρια θα είναι απαισιόδοξα (δηλαδή ψηλή τάση και συχνότητα) έτσι ώστε να λαμβάνονται υπόψη οι χειρότερες περιπτώσεις για την αποφυγή κατάρρευσης του συστήματος (system crash). Τα όρια που υπάρχουν στους επεξεργαστές και στην μνήμη σήμερα, θα ενισχυθούν με νέες πολιτικές διαχείρισης πόρων.

Η τεχνολογία UniServer θα ενσωματωθεί στον πρώτο 64-bit ARM based Server-on-Chip και θα αξιολογηθεί χρησιμοποιώντας εφαρμογές που αναπτύσσονται σε κλασσικά cloud business data-centres, καθώς και σε νέα περιβάλλοντα που είναι πιο κοντά στις πηγές των δεδομένων.

Το UniServer φιλοδοξεί να προσφέρει ένα πλήρως λειτουργικό πρωτότυπο που θα μετατρέψει τις ευκαιρίες στις αγορές των Big Data και IoT σε ωφέλημα προϊόντα που μπορούν να βελτιώσουν την καθημερινή ζωή του ανθρώπου, καθώς και να οδηγήσουν σε σημαντική οικονομική ανάπτυξη.

Η ομάδα του UniServer αποτελείται από ένα μείγμα τεχνογνωσίας διαφορετικών ομάδων. Πιο συγκεκριμένα αποτελείται από κορυφαίους προμηθευτές επεξεργαστών χαμηλής κατανάλωσης και Server-on-chip (ARM, APM) καθώς και κορυφαίες εταιρείες ανάπτυξης λογισμικού συστήματος (IBM).Επίσης, η ομάδα συμπληρώνεται από ερευνητικούς οργανισμούς και εταιρείες ανάπτυξης λογισμικού εφαρμογών.

3.4.2 Συμμετέχοντες Uniserver

Συντονιστής: Georgios Karakonstantis

Participant No	Participant organisation name	Part. Short name	Country
1 (Coordinator)	The Queen's University of Belfast	QUB	United Kingdom
2	University of Cyprus	UCY	Cyprus
3	University of Athens	UOA	Greece
4	Applied Micro Circuits Corporation Deutschland GmbH	APM	Germany
5	ARM Holdings	ARM	United Kingdom
6	IBM Ireland Ltd	IBM	Ireland
7	University of Thessaly	UTH	Greece
8	Worldsensing	WSE	Spain
9	Meritorius Audit Ltd	MER	Cyprus
10	Sparsity	SPA	Spain

Κεφάλαιο 4

4. Framework

4.1 Κίνητρο – Σκοπός

Για την επίτευξη επιστημονικής προόδου στον τομέα των υπολογιστών, είναι απαραίτητη η διεξαγωγή πειραμάτων στον υπολογιστή. Σε αυτά τα πειράματα οι ερευνητές επιθυμούν να έχουν μία ολοκληρωμένη εικόνα στο τι συμβαίνει στον υπολογιστή, ώστε να μπορούν να εξάγουν καλύτερα συμπεράσματα για τις έρευνες τους. Ακόμη, παρακολουθώντας συνεχώς την συμπεριφορά του υπολογιστή, μπορεί να εξαχθούν συμπεράσματα που δεν που δεν θα ήταν δυνατό να εξαχθούν αλλιώς, λόγω του μεγάλου όγκου των δεδομένων. Επομένως, ένα framework το οποίο θα επιτρέπει στην ερευνητική κοινότητα να παρακολουθεί την συμπεριφορά του υπολογιστή κατά την διάρκεια και μη των πειραμάτων θεωρείται απαραίτητο.

4.2 Μετρικά Παρατηρήσεων (Observation Metrics)

Για την επιστημονική παρατήρηση της συμπεριφοράς του υπολογιστή, όπως και για την επιστημονική παρατήρηση οποιουδήποτε άλλου φαινομένου, χρειαζόμαστε κάποια μετρικά που θα μπορούμε να αναδείξουμε ώστε οι παρατηρήσεις μας να έχουν επιστημονικό υπόβαθρο (π.χ. όταν θέλουμε να παρατηρήσουμε ένα κινούμενο αυτοκίνητο μπορούμε να χρησιμοποιήσουμε το S.I μετρικό της ταχύτητας - m/s). Στην περίπτωση μας χρειαζόμαστε μετρικά ώστε να μπορούμε να μετρήσουμε-παρατηρήσουμε την συμπεριφορά του υπολογιστή.

Θα χρησιμοποιήσουμε μετρικά που μας δείχνουν την κατάσταση του hardware όπως θερμοκρασία, τάση, ισχύ, συχνότητα.

Ακόμη θα χρησιμοποιήσουμε μετρικά που χρησιμοποιούνται από τους προγραμματιστές για να μετρήσουν/συγκρίνουν/βελτιώσουν/αποσφαλματώσουν την επίδοση των εφαρμογών τους. Αυτά τα μετρικά θα τα ονομάσουμε Performance Counters. Τέτοια μετρικά είναι τα instructions, cycles, branch misses κ.τ.λ.

Τέλος, λόγω του ότι στο εργαστήριο υπάρχει spectrum analyzer, χρησιμοποιήθηκε για την λήψη των μετρήσεων ηλεκτρομαγνητικής ακτινοβολίας (ΕΜ) μέσω αντένας.Η ΕΜ ακτινοβολία είναι συμπληρωματική των πιο πάνω μετρικών

Πιο κάτω μπορούμε να δούμε όλα τα μετρικά που επελέγησαν για να λαμβάνονται.

4.2.1 Λίστα Μετρικών(features)

- **Power**
  - PMD_power
  - SoCVRD(Uncore)_power
  - DIMM1_power
  - DIMM2_power
- **Temperature**
  - PMD_temp
  - SoC_temp
  - SoCVRD(Uncore)_temp
  - DIMM0_temp
  - DIMM1_temp
  - DIMM2_temp
  - DIMM3_temp
- **Voltage**
  - PMD_voltage
  - SoCVRD(Uncore)_voltage
  - DIMM1_voltage
  - DIMM2_voltage

- DIMM refresh Rate
- DIMM frequency
- L3 frequency

- **Utilization**
  - usr
  - nice
  - sys
  - iowait
  - irg
  - soft
  - steal
  - guest
  - gnice

- instructions
- cycles
- L2 accesses
- L2 misses
- L3 misses
- branch misses
- syscalls
- L2 prefetch
- exception taken
- L1 misses
- L1 TLB misses
- decode starved for instruction cycle
- op dispatch stalled cycle
- BTB misprediction
- Branch speculative executed - Indirect branch
- Branch speculative executed - Immediate branch
- L1 data TLB refill - Write
- Operation speculatively executed - Integer data processing
- Operation speculatively executed - Advanced SIMD
- Operation speculatively executed – FP

- **CPU Frequency**
  - CPU0
  - CPU1
  - CPU2
  - CPU3
  - CPU4
  - CPU5
  - CPU6
  - CPU7

Additional for X-Gene 3

- **Voltage Droop Counters**
  - 30mV cycles
  - 50mV cycles

Additional Besides Monitor

- Power from plug
- Healthlog messages
- System log messages
- EM from spectrum analyzer

Επιθυμούμε να έχουμε μία όσο το δυνατό πιο ολοκληρωμένη εικόνα για την συμπεριφορά του υπολογιστή μας για αυτό θέλουμε να λαμβάνουμε αρκετά μετρικά. Παρόλα αυτά, όσο περισσότερα μετρικά λαμβάνουμε, τόσο περισσότερο επιβαρύνεται το σύστημα μας, κάτι που θα έχει επιπτώσεις στην επίδοση των εφαρμογών που θα τρέχουν στον υπολογιστή. Οπότε έπρεπε να επιλέξουμε ανάμεσα σε όλα τα πιθανά μετρικά, μερικά που θα λαμβάνουμε και καταλήξαμε στα πιο πάνω. Τα συγκεκριμένα μετρικά επιλέγηκαν σύμφωνα με το επιστημονικό paper του University of Athens με το όνομα “Statistical Analysis of Multicore CPUs Operation in Scaled Voltage Conditions”. Επίσης συμπληρώθηκαν μετρικά που θεωρούνται σημαντικά κατόπι διαβούλευσης, και δεν αναφέρονταν στο paper. Τα επιπλέον μετρικά, είναι μετρικά που θα συμπληρώσουν το framework μας με επιπλέον μετρήσεις.

Για την παρακολούθηση της συμπεριφοράς του συστήματος μας απαιτείται η συλλογή διαφόρων μετρικών. Τα μετρικά που επιθυμούμε το framework μας να συλλέγει, είναι αυτά που αναφέρθηκαν στην σελίδα 19. Επιπλέον, χρειαζόταν να αποφασίσουμε την συχνότητα με την οποία θα λαμβάναμε τα μετρικά μας και κάποιο αποτελεσματικό τρόπο για να γνωρίζουμε την χρονική στιγμή που πήραμε κάθε μετρικό.

Για την συχνότητα που θα λαμβάνουμε τα μετρικά θεωρήθηκε ιδανική η λήψη κάθε μετρικού ανά δευτερόλεπτο. Δηλαδή σε κάθε δευτερόλεπτο επιθυμούμε να έχουμε τιμή για κάθε ένα από τα μετρικά μας. Η συγκεκριμένη συχνότητα επιλέχθηκε έπειτα από διαβούλευση. Πιστεύουμε ότι αν παίρναμε τις μετρήσεις με χαμηλότερη συχνότητα (π.χ. κάθε μετρικό ανά 2 δευτερόλεπτα) θα χάναμε πληροφορίες για την συμπεριφορά του υπολογιστή, ενώ αν παίρναμε τις μετρήσεις με χαμηλότερη συχνότητα (π.χ. 2 φορές κάθε μετρικό ανά 2 δευτερόλεπτα) θα χαλούσαμε την επίδοση του υπολογιστή σε άλλα προγράμματα, αφού ο υπολογιστής θα έχανε πολλούς πόρους προσπαθώντας να λάβει μετρήσεις αντί να απασχολείται με τα προγράμματα.

4.2.2Timestamp

Όπως αναφέρθηκε και προηγούμενος, χρειαζόμαστε κάποιο αποτελεσματικό τρόπο για να μπορούμε να αντιστοιχούμε τις μετρήσεις μας με χρονικές στιγμές. Αν απλά λαμβάνουμε συνεχείς μετρήσεις, μη μπορώντας να κατανοήσουμε πότε λήφθηκαν, θα

ήταν ανώφελο για εμάς. Δεν θα μπορούσαμε να τα αντιστοιχίσουμε με πειράματα και καταστάσεις στον υπολογιστή και δεν θα μπορούσαμε να έχουμε ακριβή συμπεράσματα και σωστή παρακολούθηση της συμπεριφοράς. Για να αντιστοιχούμε τις μετρήσεις μας με χρονικές στιγμές χρησιμοποιήσαμε το timestamp των UNIX. Το UNIX timestamp είναι ένας τρόπος για να καταγράφουμε τον χρόνο. Είναι ο αριθμός των δευτερολέπτων που έχουν περάσει από **01/01/1970 00:00:00 UTC**. Για παράδειγμα στην ημερομηνία 01/01/2019 και ώρα 00:00:00 είχαν περάσει 1546300800 δευτερόλεπτα από τις 01/01/1970 00:00:00. Επομένως το timestamp αυτής της ημερομηνίας και ώρας είναι 1546300800. Καλώντας την εντολή **"date +%s"** στους servers (και σε κάθε UNIX υπολογιστή) επιστρέφεται το timestamp της χρονικής στιγμής που καλέστηκε η εντολή. Επομένως, εμείς χρησιμοποιούμε αυτή την ιδιότητα του timestamp και σε κάθε δευτερόλεπτο, μαζί με την καταχώρηση όλων των μετρικών μας, καταχωρούμε και το timestamp, ώστε να γνωρίζουμε ότι την συγκεκριμένη χρονική στιγμή πάρθηκαν τα συγκεκριμένα μετρικά.

4.3 Framework παρακολούθησης συμπεριφοράς του συστήματος

Καταχωρούμε όλες μας τις μετρήσεις μαζί με το timestamp σε ένα συγκεκριμένο αρχείο που ονομάσαμε **monitor**. Ο τρόπος που καταχωρούμε τα δεδομένα μας είναι ο εξής:

1. Κάθε γραμμή στο αρχείο monitor αντιπροσωπεύει μία χρονική στιγμή.
2. Το timestamp και κάθε μέτρηση που λαμβάνεται, είναι σε ξεχωριστή και μοναδική στήλη στο αρχείο (π.χ. όλες οι μετρήσεις για PMD_power είναι στην δεύτερη στήλη).
3. Κάθε γραμμή του αρχείου που καταχωρείται, έχει σαν πρώτη στήλη της το timestamp της χρονικής στιγμής, ακολουθούμενη από τα υπόλοιπα μετρικά με την σειρά που φαίνονται στην σελίδα 19(όχι τα επιπλέον μετρικά).Θα ονομάσουμε την κάθε γραμμή του αρχείου **monitor vector.**

Συνεπώς, το αρχείο monitor έχει την εξής μορφή:

$Timestamp1 $PMD_power1 $SoCVRD_power1 DIMM1_power1 … (μέχρι το τελευταίο μετρικό)

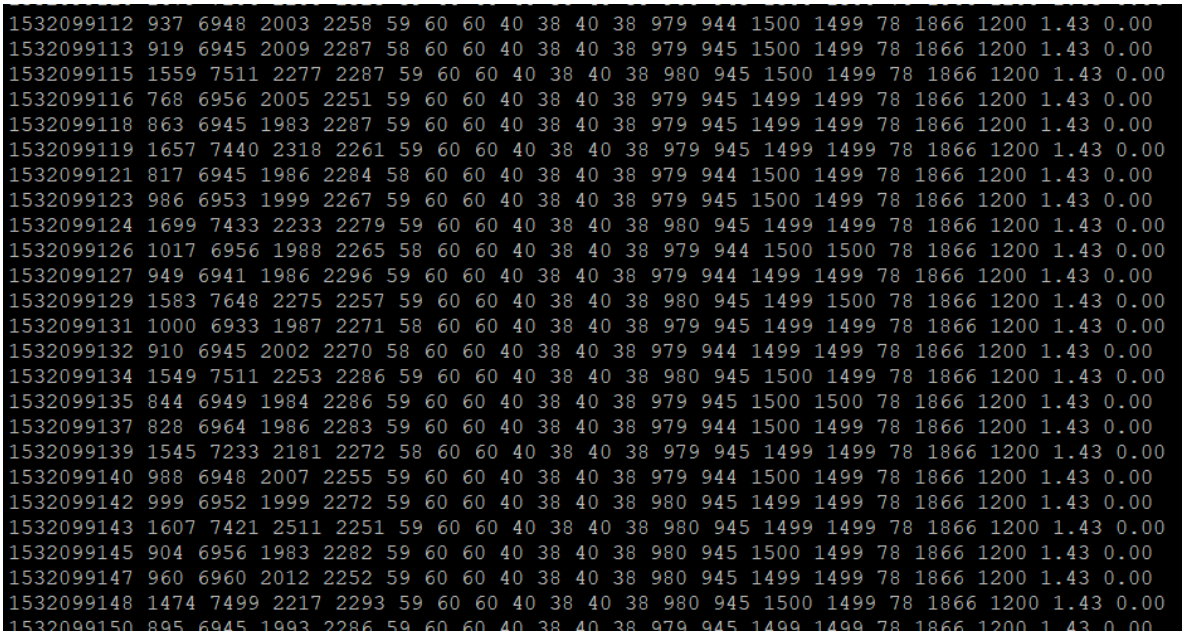
$Timestamp2 $PMD_power2 $SoCVRD_power2 DIMM1_power2 … (μέχρι το τελευταίο μετρικό)

.

.

.

Ενδεικτικά δίνουμε μία πραγματική εικόνα για το πως φαίνεται το αρχείο μας.



Θεωρούμε ότι κάθε στιγμή στον υπολογιστή είναι σημαντική και επιθυμούμε να πάρουμε όσα περισσότερα δεδομένα μπορούμε. Για αυτό το λόγο θέλουμε να λαμβάνουμε δεδομένα από την εκκίνηση (boot) του υπολογιστή. Για να το πετύχουμε αυτό, χρησιμοποιήσαμε το αρχείο rc.local στο φάκελο /etc/rc.d/ όπου οι εντολές σε αυτό το αρχείο εκτελούνται με την εκκίνηση του υπολογιστή και καλέσαμε το δικό μας script.

Με αυτό τον τρόπο κερδίσαμε:

- Το σύστημά μας να έχει γίνει πιο αυτοματοποιημένο και να μην χρειάζεται να το καλούμε κάθε φορά
- Να παίρνουμε δεδομένα για κάθε χρονική στιγμή, ακόμα και για αυτές την ώρα τις εκκίνησης του υπολογιστή.

Παίρνουμε τα δεδομένα μας από τον κώδικα μας που υπάρχει στο αρχείο Perf_counters.sh (Παράρτημα Α). Χρησιμοποιούμε τρια εργαλέια για να πάρουμε τις μετρήσεις μας:

- Perf stat- μας δίνει της performance counters μετρήσεις
- I2cget- μας δίνει μετήσεις power, tempereture, voltage,frequency, Refresh Rate
- Mpstat – μας δίνει μετρήσεις για utilization

4.4 Framework για καταχώρηση πειραμάτων

Πιο πάνω αναλύσαμε πως παίρνουμε συνεχώς μετρικά που θα μας βοηθήσουν στην παρακολούθηση της συμπεριφοράς του συστήματος. Δεν έχουμε ακόμη όμως κάποιο συστηματικό τρόπο για να καταχωρούμε τα πειράματά μας, δηλαδή να γνωρίζουμε ανά πάσα στιγμή στον υπολογιστή ποια εφαρμογή/πείραμα (benchmark ή application) έτρεχε. Αυτό είναι σημαντικό ώστε να μπορούμε να αντιστοιχίσουμε τα μετρικά που παίρνουμε (εξηγήσαμε πιο πάνω πως) με τα πειράματά μας. Γνωρίζοντας ποια πειράματα έτρεχαν μία χρονική στιγμή και γνωρίζοντας τις τιμές των μετρικών αυτή την στιγμή, μπορούμε να είμαστε σίγουροι ότι η συμπεριφορά του υπολογιστή την συγκεκριμένη χρονική στιγμή, είναι απόρροια του πειράματος που έτρεξε. Επομένως μπορούμε να εξαγάγουμε ορθά συμπεράσματα για την συμπεριφορά του υπολογιστή υπό τις συνθήκες του συγκεκριμένου πειράματος.

Για την καταχώριση των πειραμάτων μας θα χρειαστεί να απευθυνθούμε και πάλι στο timestamp που εξηγήσαμε προηγουμένως (σελίδα 20). Χρειαζόμαστε το timestamp ώστε να καταχωρήσουμε την χρονική στιγμή που ξεκίνησε και την χρονική στιγμή που τελείωσε το πείραμά μας.

Ο κώδικας που υπάρχει στο αρχείο Script_Wrapper.sh(Παράρτημα Β), είναι υπεύθυνος για την καταγραφή των πειραμάτων. Στην εκτέλεση του αρχείου Script_Wrapper.sh, η

πρώτη παράμετρος που δίνεται είναι το πρόγραμμα/ benchmark που θέλουμε να εκτελεστεί, ακολουθούμενη από τις παραμέτρους του benchmark. Το αρχείο Script_Wrapper.sh καταχωρεί το πείραμα ως εξής:

1. Κάνει αντίτυπο(copy) του benchmark που θα τρέξει συμπληρώνοντας το στην τελευταία γραμμή με τις παραμέτρους που θα τρέξει (θέλουμε να γνωρίζουμε τις παραμέτρους που τρέχουμε ένα benchmark)
2. Μετονομάζει το αντίτυπο σε $timestampStart, όπου $ timestampStart η χρονική στιγμή που το benchmark είναι έτοιμο να τρέξει.
3. Αφού το benchmark έχει τελειώσει, μετονομάζει ξανά το αντίτυπο σε $timestampStart_$timestampEnd, όπου $timestampEnd η χρονική στιγμή που το benchmark είναι τελείωσε.

Με αυτό τον τρόπο ξέρουμε ανά πάσα στιγμή τι πείραμα έτρεχε, ποιες ήταν οι παράμετροι του, πότε ξεκίνησε και πότε τελείωσε.

ΣΗΜΑΝΤΙΚΟ: Αν βρούμε ένα καταγεγραμμένο πείραμα της μορφής $timestampStart και όχι της μορφής $timestampStart_$timestampEnd, ξέρουμε ότι δεν τελείωσε ποτέ και επομένως είχε προκληθεί crash σαν έτρεχε.

Έτσι ξέροντας σε κάθε timestamp ποιο benchmark ‘έτρεχε, μπορούμε να το συνδέσουμε με τις μετρήσεις που έχουμε στο monitor και να ξέρουμε για κάθε χρονική στιγμή του benchmark τις τιμές των μετρικών.

Στην περίοδο που χρησιμοποιούμε το framework, έχουμε τρέξει και καταγράψει πολλά διαφορετικά πειράματα. Ενδεικτικά θα αναφέρουμε μερικές από τις ομάδες πειραμάτων που τρέξαμε:

- Polaris
- WSE
- SPEC2006
- SPEC2017
- Parsec
- NPB
- Πειράματα idle (εντολή sleep)

4.5 Συμπληρωματικά εργαλεία για το framework

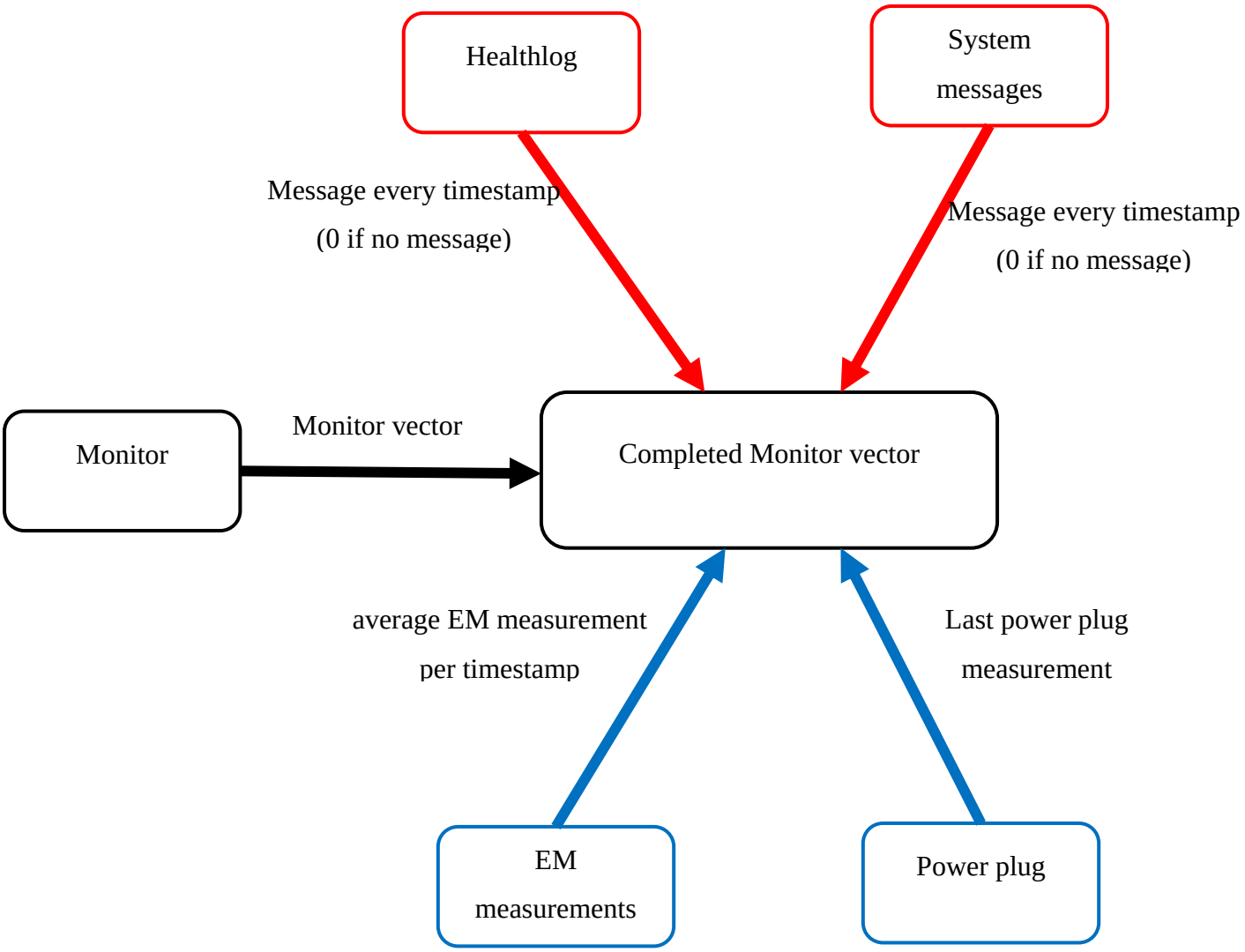
Θα αναφέρουμε μερικά εργαλεία που έχουμε υλοποιήσει, που συμπληρώνουν την λειτουργία του framework που περιγράψαμε.

Κατά αρχήν, λόγω του ότι το timestamp δεν είναι εύχρηστος τρόπος αναγνώρισης του χρόνου από τους ανθρώπους, χρειαζόμαστε ένα αρχείο που να μπορεί να μετατρέπει το timestamp σε χρόνο στη μορφή του Γρηγοριανού Ημερολογίου και αντίστροφα. Επιπλέον σαν επέκταση αυτού του εργαλείου, θέλουμε δίνοντας ημερομηνία και ώρα, να μας επιστρέφεται το πείραμα που έτρεχε εκείνη την στιγμή στον server.Έτσι το framework είναι πιο λειτουργικό.

Αφού καταγράφουμε τα ολοκληρωμένα πειράματά μας σε μορφή 2 timestamps, θέλουμε ένα εργαλείο όπου δίνοντας του ένα καταγεγραμμένο ολοκληρωμένο πείραμα (είτε δύο timestamps) να μας επιστρέφει όλα τα monitor vectors αυτού του πειράματος. Αυτό το εργαλείο ολοκληρώνει περεταίρω το framework, αφού κάνει την αντιστοιχία πείραμα – μετρικών αυτόματη.

Όπως προαναφέραμε, πέραν του monitor, παίρνουμε μετρήσεις και από άλλες πηγές· **healthlog messages**, **system log messages**, **EM measurements** και **power plug**. Μία από τις προκλήσεις που είχαμε, ήταν στο να μπορέσουμε να συνδυάσουμε αυτές τις μετρήσεις, δημιουργώντας ένα πιο ολοκληρωμένο vector. Για να μπορέσουμε να συνδυάσουμε τις μετρήσεις μας, πρέπει σε κάθε αρχείο που έχουμε αποθηκευμένες μετρήσεις (monitor file, file that has EM measurements, Healthlog file, system messages file) να πάρουμε κάθε Timestamp και να το συνδυάσουμε με τα αντίστοιχα timestamps των άλλων files. Ταυτόχρονα, οι μετρήσεις που παίρναμε σε κάθε περίπτωση (κάθε file), δεν παραγόντουσαν με την ίδια συχνότητα. Αυτό έδινε επιπλέον δυσκολία στο πρόβλημα αφού έπρεπε να αποφασίσουμε ποιες τιμές θα επιλέγαμε για την ολοκλήρωση του monitor vector.Στην περίπτωση των EM μετρήσεων, υπήρχαν περισσότερες από μία μετρήσεις ανά δευτερόλεπτο. Σαν συμβιβασμό, πήραμε τον μέσο όρο των μετρήσεων που αντιστοιχούσαν σε εκείνο το timestamp. Αντίστοιχα, στις περιπτώσεις των Healthlog και system messages, θα καταγράφαμε τιμή 0 όταν δεν υπήρχε κάποιο μήνυμα σε κάποιο timestamp. Όταν υπάρχει μήνυμα, θα καταγράφουμε το μήνυμα σαν ένας αριθμός, που θα είναι ο τύπος του μηνύματος κωδικοποιημένος (π.χ. για μήνυμα τύπου Χ θα

καταγράφουμε τον αριθμό 1, ενώ για μήνυμα τύπου Y τον αριθμό 2 κ.τ.λ.).Στην περίπτωση του plug, στα monitor vectors που υπολείπονται power plug τιμής, θα καταγράφουμε ως τιμή, την τελευταία τιμή που καταχωρήθηκε. Πιο κάτω παρουσιάζεται διαγραμματικά η επικοινωνία.



Τέλος στα πειράματα τα οποία δεν έχουν ολοκληρωθεί (δηλαδή η μορφή τους είναι μόνο ένα timestamp), επιθυμούμε να βρούμε το σημείο στο οποίο έχει γίνει το crash. Για αυτό θα χρειαστούμε μία επιπλέον ιδιότητα που περιγράφεται στην σελίδα 40.

Πιο κάτω, παραθέτουμε ένα πίνακα με τα συμπληρωματικά αρχεία / queries που υλοποιήσαμε για το framework και μία επεξήγησή της χρήσης τους.

Αρχείο	**Περιγραφή**
Aver_stat.sh	Calculate the average of a feature in a file
Combine_monitors.sh	Creates the complete monitor vector
Convert_Timestamp.sh	Converts date to timestamp
Distribution_of_timesDiff.sh	Calculate Deltas of timestamps in monitor
EM_subMonitor.sh	Creates a subMonitor for EM measures
Find_crash.sh	Find crash point on a file
Gnuplot_script.sh	Plot a feature of a file
Perf_counters.sh	Take measurements to Monitor
Script_Wrapper.sh	Register Experiments
subMonitor.sh	Create a subMonitor using 2 timestamps

Κεφάλαιο 5

5. Polaris Characterization

5.1 Όρος characterization

Όταν θα χρησιμοποιήσουμε μία εφαρμογή κάπου επιστημονικά, πολλές φορές μας ζητείται να την χαρακτηρίσουμε (να κάνουμε characterization). Το characterization μιας εφαρμογής είναι όταν προσπαθούμε να εξερευνήσουμε την εφαρμογή μας περισσότερο, ώστε να κατανοήσουμε καλύτερα μερικές όψεις τις. Μπορούμε να την εξερευνήσουμε για να βρούμε τον βέλτιστο δυνατό τρόπο για να την τρέχουμε, την κατανάλωση της εφαρμογής πως επιδρά η εφαρμογή στο σύστημα κ.τ.λ.

5.2 Υλοποίηση characterization

Για τους σκοπούς του UniServer ανατέθηκε στο εργαστήριο μας το characterization της Polaris εφαρμογής που αναλύσαμε πιο πάνω. Χρησιμοποιήσαμε το framework που υλοποιήσαμε για να χαρακτηρίσουμε την εφαρμογή μας αφού μας δίνει όλα τα απαραίτητα εργαλεία και αυτοματισμούς. Για τους σκοπούς του UniServer Project, προτιμήθηκε το characterization της εφαρμογής μέσω Virtual Machine (η VM προσομοιώνει την λειτουργία ενός υπολογιστικού συστήματος). Χρησιμοποιήσαμε KVM Virtual Machine για το characterization της Polaris εφαρμογής που έγινε στον X-Gene 2 Server.

Για τους σκοπούς του characterization επιθυμούσαμε να ανακαλύψουμε τα εξής για το Polaris:

1. Τον καλύτερο δυνατό τρόπο να τρέχουμε την εφαρμογή μας. Επιθυμούμε όσο το δυνατό περισσότερα στιγμιότυπα **(instances)** της εφαρμογής να τρέχουν παράλληλα **χωρίς** όμως να παραβιάζεται το QoS της εφαρμογής.
2. Την ελάχιστη ασφαλή τάση του επεξεργαστή (θα την ονομάσουμε **Vmin**) στην οποία η εφαρμογή μας έχει την δυνατότητα να τρέξει χωρίς να προκαλέσει crash.
3. Την κατανάλωση της εφαρμογής μας τόσο σε nominal συνθήκες, όσο και σε συνθήκες Vmin. Από εδώ μπορούμε να βρούμε και την εξοικονόμηση ενέργειας όταν τρέχουμε την εφαρμογή μας σε Vmin καταστάσεις.

5.3 Διαδικασία

Για τον χαρακτηρισμό της εφαρμογής αρχικά επιθυμούσαμε να βρούμε τον καλύτερο δυνατό τρόπο για να τρέχουμε την εφαρμογή. Λόγω του ότι ο X-Gene 2 Server έχει 8 φυσικούς πυρήνες, ξεκινήσαμε το characterization της εφαρμογής από 1 μέχρι 8 instances (εννοώντας 8 ταυτόχρονα instances). Αν η εφαρμογή θα εξακολουθούσε να βρισκόταν στα όρια του QoS στα 8 instances, τότε θα επιχειρούσαμε για ακόμη περισσότερα (αυτό δεν συνέβη και η το characterization της εφαρμογής έμεινε μέχρι τα 8 instances). Οι παράμετροι που είχαμε να εξερευνήσουμε στο characterization ήταν:

- Τον αριθμό των instances
- Τον αριθμό των virtual cores της VM
- Οι φυσικοί πυρήνες στους οποίους “τρέχουν” (κάνουν map) οι virtual cores

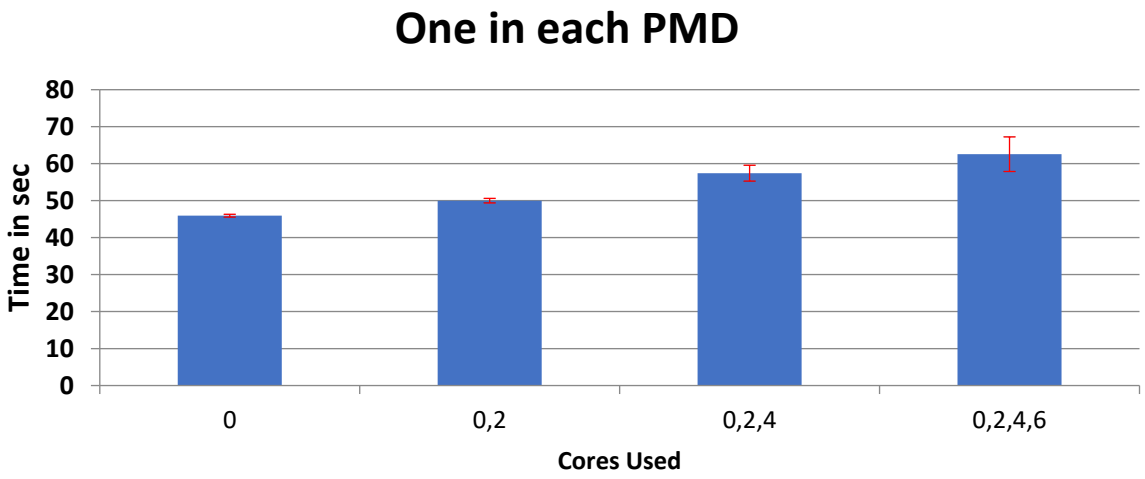
Αφού βρούμε τον τρόπο με τον οποίο θα τρέχουμε την εφαρμογή, θα βρούμε το Vmin της εφαρμογής για τον τρόπο που προτείναμε να τρέχει. Ακολούθως, βασισμένοι στο Vmin που βρήκαμε, θα βρούμε την κατανάλωση της εφαρμογής σε nominal και σε Vmin καταστάσεις και θα αναλύσουμε την εξοικονόμηση ενέργειας που πετυχαίνουμε τρέχοντας την εφαρμογή σε καταστάσεις Vmin.

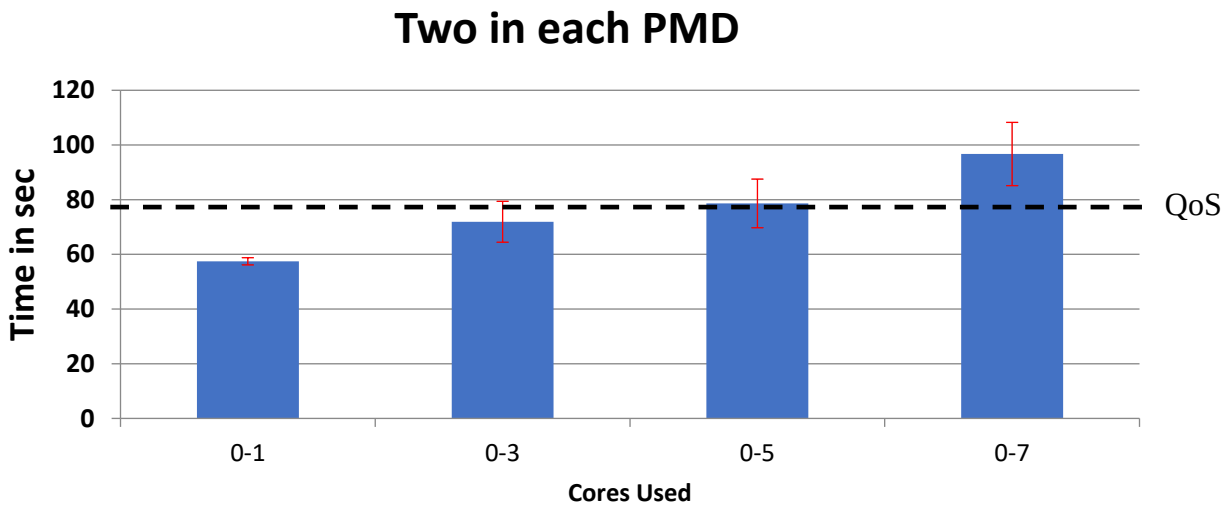
5.4 Αποτελέσματα – συμπεράσματα

5.4.1 Προτεινόμενος τρόπος να τρέχει η εφαρμογή

Εξετάσαμε όλες τις παραμέτρους που αναφέρθηκαν πιο πάνω. Αρχικά παρατηρήσαμε ότι ο καλύτερος αριθμός virtual cores για κάθε περίπτωση, ήταν ίσος με τον αριθμό των instances. Επομένως για κάθε μας πείραμα χρησιμοποιούμε των ίδιο αριθμό virtual cores με instances.

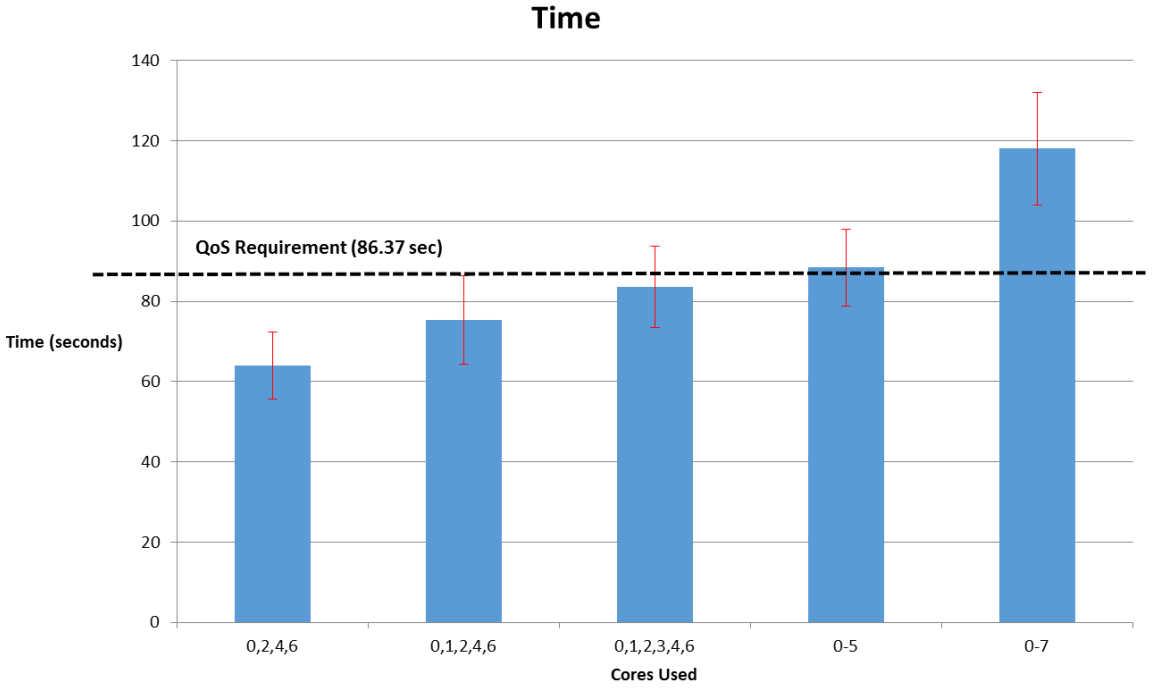
Ακολούθως αναλύσαμε αν το πού τρέχουν οι virtual cores (και επομένως και τα instances) επηρεάζει τον χρόνο εκτέλεσης της εφαρμογής.





Από τις πιο πάνω γραφικές συνειδητοποιούμε ότι τα instances τα οποία εκτελούνται σε φυσικούς πυρήνες που βρίσκονται στο ίδιο PMD έχουν πιο χαμηλή επίδοση (ψηλότερο χρόνο εκτέλεσης). Ο λόγος που αυτό συμβαίνει είναι λόγω της L2 cache που είναι κοινή ανάμεσα σε φυσικούς πυρήνες του ιδίου PMD. Η L2 cache έχει περισσότερα misses όταν τρέχουν 2 Polaris instances στο ίδιο PMD, κάτι που επηρεάζει προς το αρνητικό την επίδοση.

Ακολούθως αναλύσαμε διαφορετικό αριθμό instances. Επιθυμούμε να έχουμε όσο το δυνατό περισσότερο αριθμό instances να τρέχουν παράλληλα χωρίς να παραβιάζουμε το QoS. Τα αποτελέσματα της ανάλυσής μας φαίνονται στην πιο κάτω γραφική.



Όπως μπορούμε να παρατηρήσουμε η καλύτερη επιλογή ,στην οποία παραμένουμε στο QoS, και στην οποία και καταλήξαμε, είναι με 6 instances στους πυρήνες 0,1,2,3,4,6 (οι πυρήνες 0,1 και 2,3 βρίσκονται στα ίδια PMD ενώ οι πυρήνες 4,6 σε διαφορετικά).

Επομένως η τελική μας πρόταση περιλαμβάνει τα εξής:

- **6 virtual cores στο VM**
- **6 instances με ένα instance σε κάθε virtual core**
- **Οι 6 virtual cores "τρέχουν" στους πυρήνες 0,1,2,3,4,6**

5.4.2 Εύρεση Vmin

Αφού αναλύσαμε και βρήκαμε τον ιδανικό τρόπο που πρέπει να τρέχει η εφαρμογή, θα επιχειρήσουμε να βρούμε το Vmin της εφαρμογής. **Σαν κανόνα θέσαμε ότι ένα ασφαλές Vmin είναι 20 millivolts μεγαλύτερο από το μέγιστο voltage στο οποίο η εφαρμογή μας παρουσιάζει crash.**

Για παράδειγμα εάν σε 3 διαφορετικά πειράματα η εφαρμογή παρουσιάζει crash στις τιμές:

- 920, 900, 910

τότε το Vmin μας είναι:

- MAX(920,900,910)=920 +20 = **940**

Ο κανόνας των 20 millivolts τέθηκε κατόπι διαβούλευσης. Τα 20 επιπλέον millivolts προσθέτουν ασφάλεια στο ότι η εφαρμογή μας δεν θα παρουσιάσει crash, αλλά ταυτοχρόνως δεν αυξάνουν πολύ την τάση (voltage).

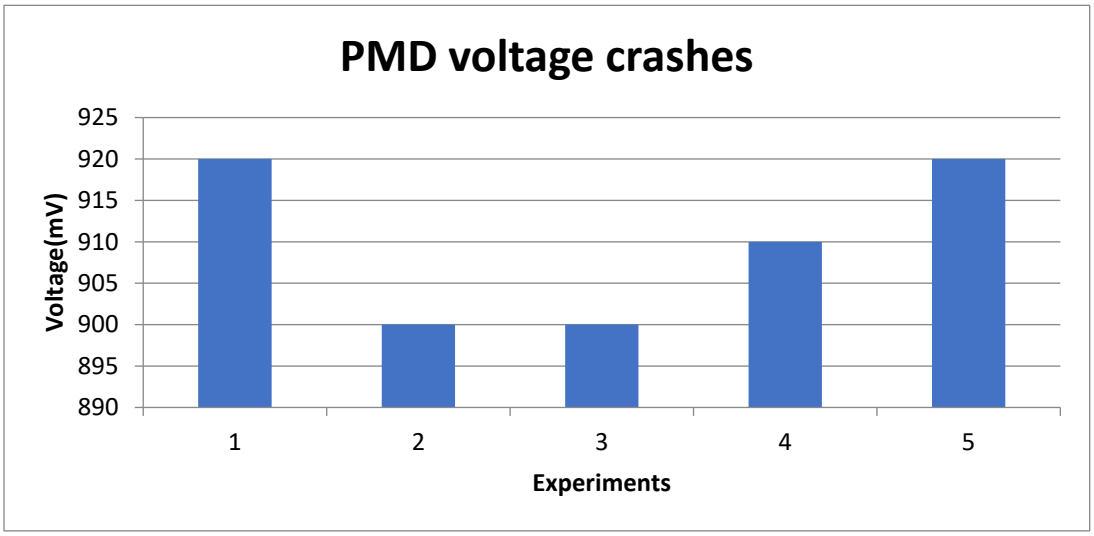
Για να βρούμε το Vmin της εφαρμογής ακολουθήσαμε την εξής διαδικασία:

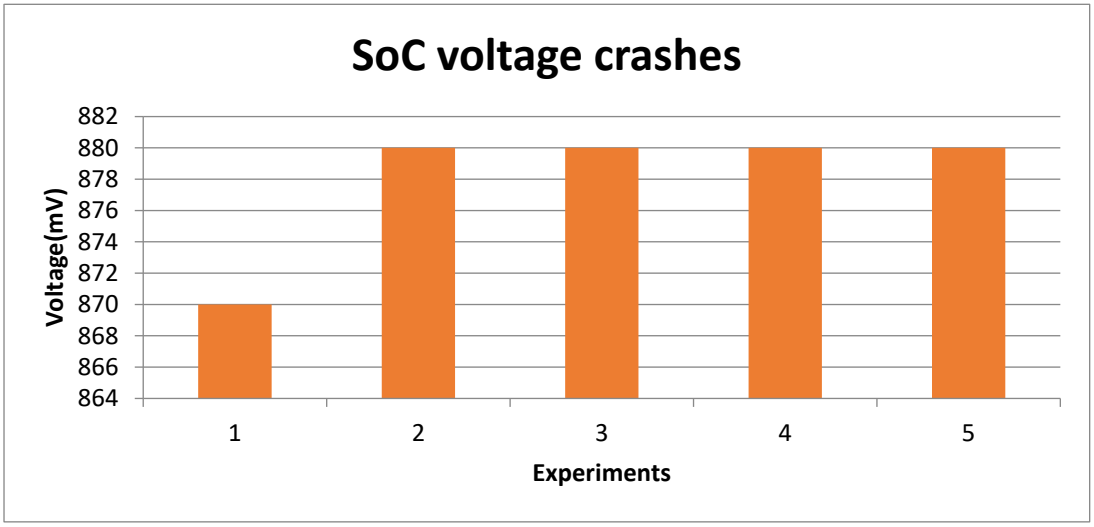
1. Ξεκινήσαμε να τρέχουμε την εφαρμογή σε Nominal συνθήκες (PMD_voltage=980millivolts , SoC_voltage=950millivolts).
2. Κατεβάσαμε το voltage των πυρήνων του επεξεργαστή (PMD_voltage) κατά 10 millivolts.
3. Τρέξαμε ξανά την εφαρμογή με το νέο voltage.

4. Αν η εφαρμογή παρουσίαζε crash στο καινούργιο voltage, καταγράφαμε το voltage και ξεκινούσαμε ξανά από το Βήμα 1. Αν η εφαρμογή δεν παρουσίαζε crash στο καινούργιο voltage επαναλαμβάναμε τα βήματα 2 και 4.
5. Επαναλαμβάνουμε την διαδικασία μέχρι να καταγράψουμε 10 voltage crashes.
6. Ακολούθως υπολογίζουμε το Vmin με την φόρμουλα που αναλύσαμε πιο πάνω, δηλαδή Vmin_PMD=MAX(Voltage_crashes[]) +20

Ακολουθούμε τα ίδια βήματα αλλά αντί για PMD_voltage, αλλάζουμε το SoC_voltage και αυτό θα μας δώσει το Vmin_SoC_voltage. Με αυτά τα δύο ξέρουμε την ελάχιστη ασφαλή τάση στην οποία πρέπει να τρέχει ο επεξεργαστής, τόσο σε επίπεδο PMD, όσο και σε επίπεδο SoC.

Ακολουθήσαμε λοιπόν την διαδικασία που περιγράψαμε πιο πάνω και καταλήξαμε στα εξής PMD_voltage_crashes και SoC_voltage _crashes.





Επομένως σύμφωνα με την πιο πάνω γραφική και την φόρμουλα για υπολογισμό του Vmin έχουμε ότι Vmin_PMD = 940 και Vmin_SoC = 900.

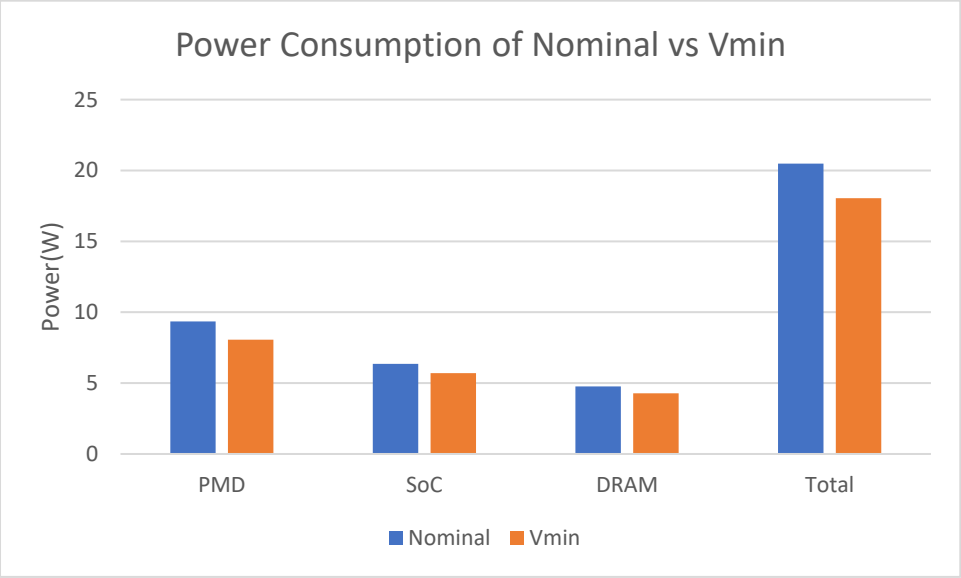
Τα πιο πάνω συμπληρώνονται με μελέτες του UniServer Project για τον υπολογισμό Vmin_DIMM και Max_Refresh Rate. Από το UniServer Project γνωρίζουμε ότι Vmin_DIMM =1428 και Max_Refresh Rate=2783.

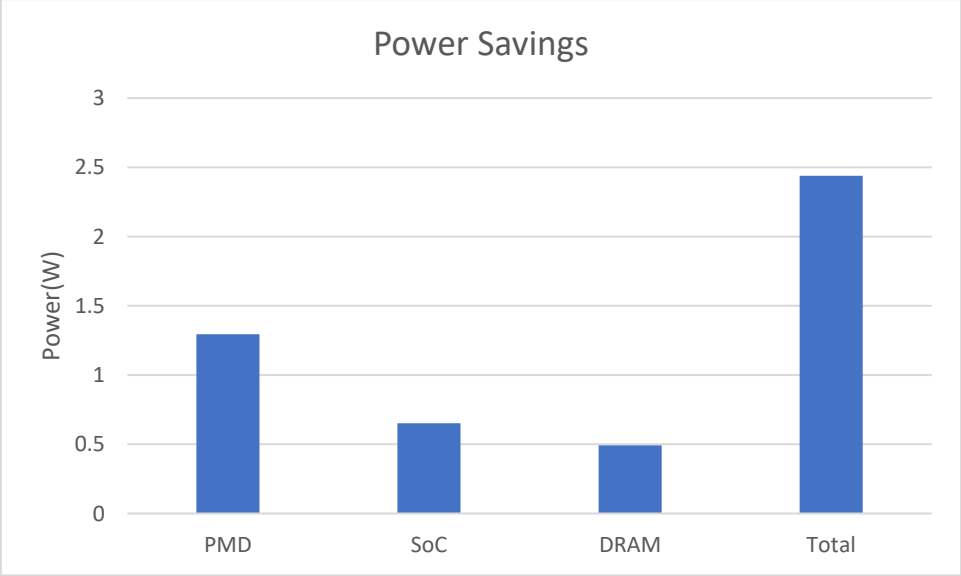
Έτσι έχουμε ότι ο καλύτερος τρόπος για να τρέξεις τη εφαρμογή για μειωμένη κατανάλωση είναι:

- Vmin_PMD = 940
- Vmin_SoC = 900
- Vmin_DIMM =1428
- Max_Refresh_Rate=2783

5.4.3 Energy Savings of Nominal vs Vmin

Σύμφωνα με το Vmin της εφαρμογής που βρήκαμε, τρέχουμε την εφαρμογή μας στις nominal συνθήκες της (V_PMD = 980 V_SoC = 950V_DIMM =1500, Refresh_Rate=78) και ακολούθως στις Vmin συνθήκες της (Vmin_PMD = 940,Vmin_SoC = 900, Vmin_DIMM =1428, Max_Refresh_Rate=2783) παρατηρώντας κάθε φορά την κατανάλωση ενέργειας της εφαρμογής. Πιο κάτω παρουσιάζονται τα αποτελέσματά μας.





Επομένως η συνολική εξοικονόμηση ενέργειας που έχουμε με το Vmin της εφαρμογής μας συγκριτικά με τις nominal συνθήκες είναι **2.5 Watts.**

Κεφάλαιο 6

6. Συμπεράσματα χρήσης του framework

6.1 Παρατηρήσεις

Το framework που αναπτύξαμε λάμβανε δεδομένα για περίοδο 6 μηνών (με ενδιάμεσες διακοπές). Σε αυτή την περίοδο, συλλέγηκαν συνολικά δεδομένα μεγέθους:

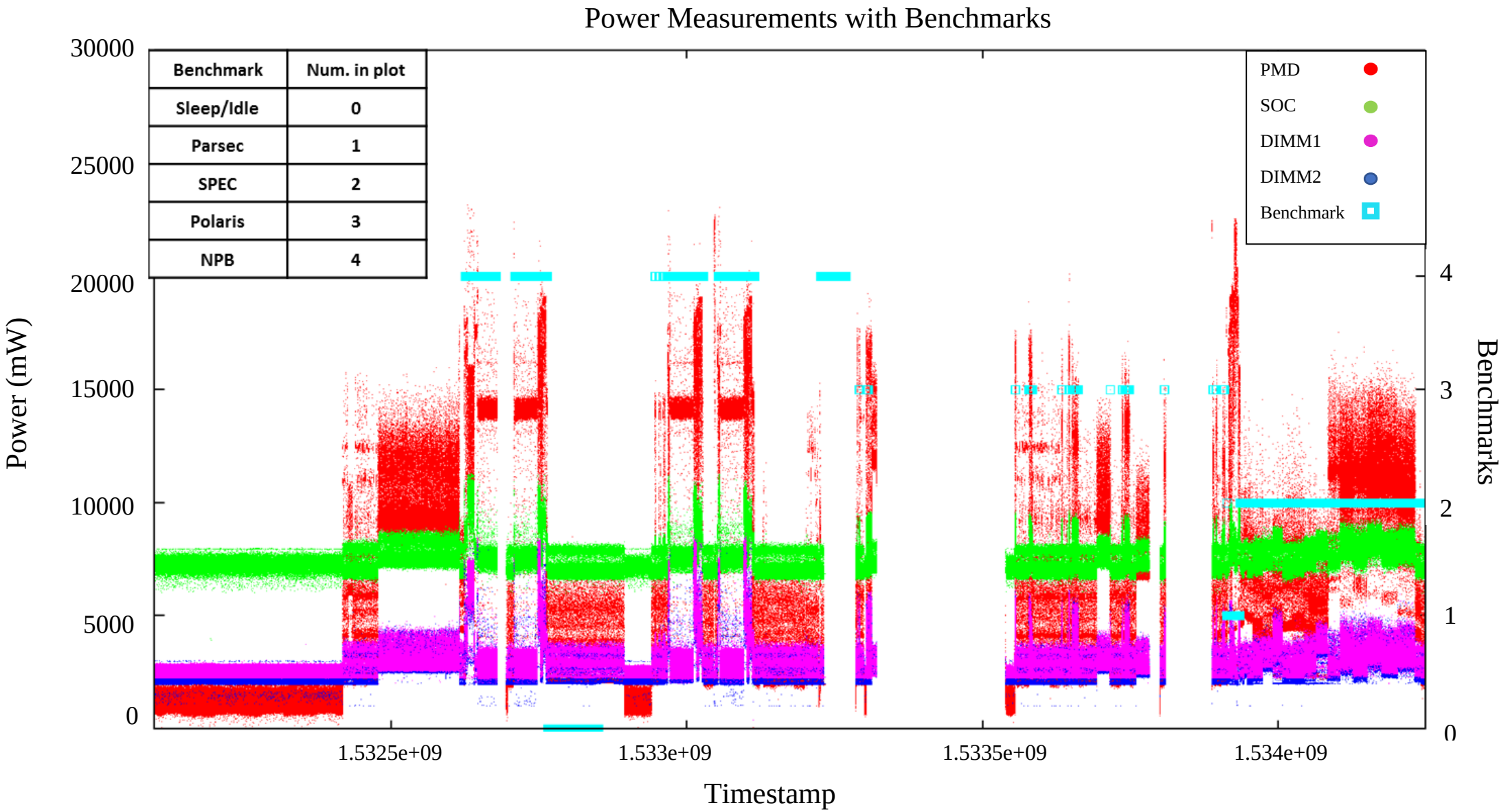
- 2GB για τον X-Gene 2 – 5M monitor vectors
- 4GB για τον X-Gene 3 – 8M monitor vectors

Και έτρεξαν 2882 καταγεγραμμένα πειράματα στον X-Gene 2 και 6125 καταγεγραμμένα πειράματα στον X-Gene 3.

Από τα όλα δεδομένα που λάβαμε, εξαγάγαμε διάφορες παρατηρήσεις γενικά για την συμπεριφορά του υπολογιστή και του framework και αναδείξαμε μερικές ανωμαλίες του συστήματος.

6.1.1 Γενικές Παρατηρήσεις

Από το framework που αναπτύξαμε και με των μεγάλο όγκο των δεδομένων που έχουμε, μπορούμε να τα χρησιμοποιήσουμε για να βρούμε τις κατανομές μετρικών και να εξαγάγουμε χρήσιμα συμπεράσματα από αυτές. Για παράδειγμα η πιο κάτω γραφική, μας δείχνει τις τιμές των μετρικών για κατανάλωση ισχύς (Power) για 1.5 μήνα, αντιστοιχώντας τις μετρήσεις με τα benchmarks που έτρεχαν εκείνη την χρονική στιγμή.


Power Measurements with Benchmarks
Benchmark | Num. in plot
Sleep/Idle | 0
Parsec | 1
SPEC | 2
Polaris | 3
NPB | 4
PMD
SOC
DIMM1
DIMM2
Benchmark
Power (mW)
30000
25000
20000
15000
10000
5000
0
Benchmarks
4
3
2
1
0
1.5325e+09
1.533e+09
1.5335e+09
1.534e+09
Timestamp

Επεξήγηση γραφικής

- Στην πιο πάνω γραφική για κάθε ένα από τα καταγεγραμμένα timestamps στην περίοδο του 1.5 μήνα υπάρχει **σημείο** για power PMD(κόκκινο), SOC(πράσινο), DIMM1(ροζ) και DIMM2(μπλε). Για κάθε timestamp των καταγεγραμμένων πειραμάτων, υπάρχει **σημείο** για το ποιο benchmark έτρεχε (cyan). ΠΡΟΣΟΧΗ: Τα σημεία είναι πολύ πυκνά και μπορεί να φανούν σαν ευθείες.
- Η πιο πάνω γραφική έχει στον x-άξονά τα timestamps (χρόνος).
- Υπάρχουν 2 y-άξονες (δεξιά και αριστερά της γραφικής).
- Ο αριστερά y-άξονας μετράει Power.Οι τιμές των PMD, SOC,DIMM1και DIMM2 αντιστοιχούνται στον αριστερά y-άξονα.
- Στον δεξιά y-άξονα αντιστοιχούνται τα benchmarks. Τα benchmarks είναι 5 (αντιστοιχούνται με τους αριθμούς 0-4) Για να δούμε την αντιστοιχία benchmarks με αριθμών, μπορούμε να δούμε τον πίνακα πάνω αριστερά της γραφικής.

Για καλύτερη κατανόηση δίνουμε ένα παράδειγμα.

Στο timestamp≈1.533e+0.9 έτρεχε το benchmark=4=NPB.Οι power μετρήσεις(PMD, SOC,DIMM1και DIMM2) δεν είναι εντελώς ξεκάθαρες μιας και τα σημεία είναι πολύ πυκνά.

Έτσι από εδώ μπορούν να εξαχθούν χρήσιμα συμπεράσματα όπως ποιο benchmark έχει την περισσότερη κατανάλωση, ποιες είναι οι ακραίες τιμές της κατανομής και πού παρουσιάζονται κ.α. Με πολλά δεδομένα μπορεί να γίνει μία Big Data ανάλυση για να εξαχθούν ακόμη περισσότερα συμπεράσματα.

6.1.2 Crashes

Τρέχοντας διάφορα benchmarks με το framework στον X-Gene 2 παρουσιάστηκαν 7 crashes σε nominal συνθήκες του επεξεργαστή. Αυτά τα crashes παρουσιάστηκαν στα εξής πειράματα:

1. SPEC2017-cactus
2. SPEC2006-gobmk
3. SPEC2006-mcf

4. SPEC2006-tonto
5. SPEC2006-bwaves
6. 2 crashes στην εφαρμογή Polaris

Αυτά τα crashes υπήρξαν η αφορμή για την ανάλυση μας για crash που αναλύουμε στην σελίδα 49.

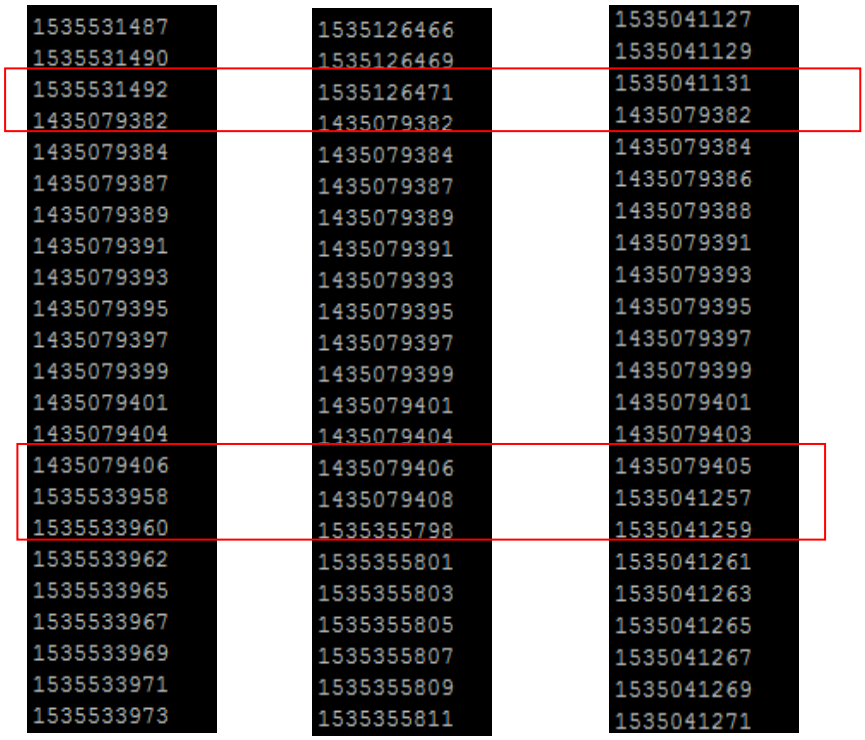
6.2 Ανωμαλίες

Με όλα τα δεδομένα που συλλέξαμε, παρατηρήσαμε ότι υπήρξαν κάποιες ανωμαλίες. Συγκεκριμένα, κάποιες τιμές του timestamp παρουσιάζονταν πολλαπλές φορές και σε διαφορετικές χρονικές περιόδους, κάτι που γνωρίζουμε ότι δεν μπορεί να συμβεί αφού το κάθε timestamp είναι μοναδικό για κάθε δευτερόλεπτο. Ακόμη, κάποιες τιμές μετρικών ήταν εξαιρετικά μεγάλες που δεν θα μπορούσαν να αντιπροσώπευαν πραγματικά τι συμβαίνει στον υπολογιστή. Προσπαθήσαμε να ανιχνεύσουμε πότε και τον λόγο για τον οποίο συμβαίνουν αυτές οι ανωμαλίες.

6.2.1 Ανωμαλία timestamp

Όπως αναλύσαμε και στην σελίδα 20 το timestamp είναι μονάδα μέτρησης χρόνου και είναι ο αριθμός των δευτερολέπτων που έχουν περάσει από **01/01/1970 00:00:00 UTC.** Συνεπώς, 2 timestamps που πάρθηκαν με διαφορά χρόνου μεγαλύτερη ή ίση του ενός δευτερολέπτου δεν μπορούν να έχουν το ίδιο timestamp. Παρόλα αυτά, παρατηρήσαμε ότι βρίσκαμε πολλαπλά timestamps με την ίδια τιμή που πάρθηκαν σε διαφορετικές χρονικές περιόδους, κάτι που μας φάνηκε ανωμαλία. Αναλύοντας περισσότερο πότε παρουσιάζονται οι τιμές αυτές, βρήκαμε ότι παρουσιάζονται όταν ξεκινά (boot) ο server. Συγκεκριμένα, σε κάθε boot του server, για μερικά δευτερόλεπτα, τα timestamp που παίρνονται είναι timestamps που αντιστοιχούν στο γρηγοριανό ημερολόγιο στις **23/06/2015**. Η ημερομηνία αυτή πιστεύουμε ότι είναι η ημερομηνία που φτιάχτηκε ο server. Μετά το booting process, για μερικά δευτερόλεπτα παρουσιάζονται τιμές αυτής

της ημερομηνίας (δεν είναι σταθερό το πόσα δευτερόλεπτα), και ακολούθως οι τιμές που λαμβάνονται είναι οι σωστές. Πιο κάτω είναι μία ενδεικτική εικόνα.



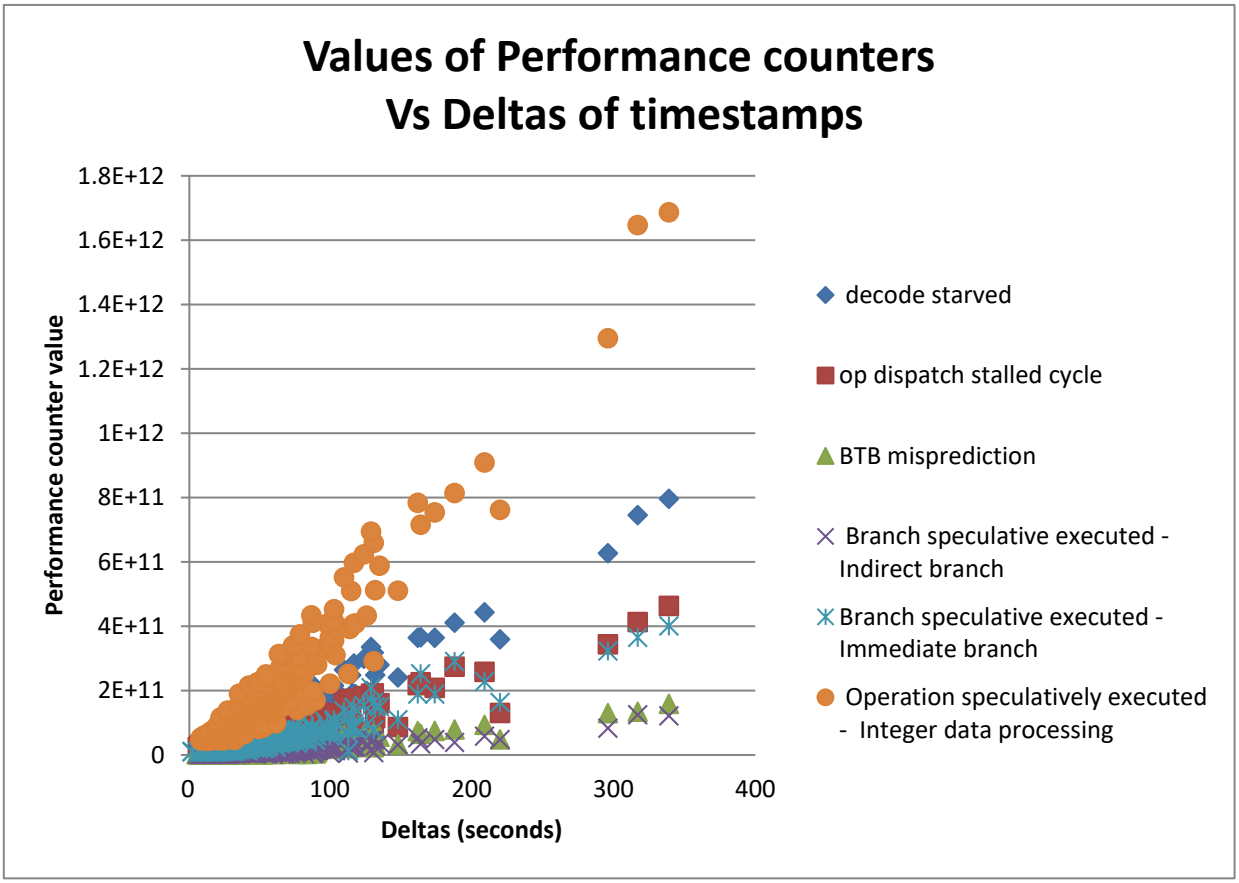
Βρίσκοντας αυτή την ανωμαλία, αποκτήσαμε μία επιπλέον δυνατότητα. Μπορούμε πλέον πολύ εύκολα να ανιχνεύουμε πότε έκανε boot η μηχανή. Αρκεί να βρούμε τις περιόδους που παρουσιάζονται τιμές της ημερομηνίας 23/06/2015.

6.2.2 Ανωμαλία μεγάλων μετρικών

Παρατηρώντας τα μετρικά που συλλέξαμε, βρήκαμε ότι κάποιες τιμές μετρικών ήταν εξαιρετικά μεγάλες για να αντιπροσωπεύουν μετρικά 1 δευτερολέπτου. Θέλαμε να βρούμε γιατί παρουσιάζεται αυτή η ανωμαλία.

Για να πάρουμε μετρικά από τον X-Gene 2 Server χρησιμοποιήσαμε την εντολή **perf stat sleep 1**, που παίρνει μετρικά για ένα δευτερόλεπτο. Έπειτα από αυτό το δευτερόλεπτο, έχουμε τις τιμές των μετρικών. Βρήκαμε ότι η ανωμαλία των μεγάλων μετρήσεων παρουσιάζεται όταν, για κάποιο λόγο, η εντολή **perf stat sleep 1** διαρκεί πολλά δευτερόλεπτα και όχι ένα. Γνωρίζουμε ότι παίρνει πολλά δευτερόλεπτα με το να

κοιτάξουμε την επόμενο καταγεγραμμένο timestamp. Αν το επόμενο καταγεγραμμένο timestamp έχει X δευτερόλεπτα διαφορά από το προηγούμενο, τότε οι μετρήσεις που βλέπουμε είναι X δευτερολέπτων. Πιο κάτω είναι μία γραφική των διαφορών (deltas) των timestamps συγκριτικά με τις τιμές κάποιων μετρικών.



Για να επιλύσουμε αυτό το πρόβλημα μπορούμε να διαγράψουμε αυτές τις μετρήσεις, αφού δεν τις θεωρούμε έγκυρες και αξιόπιστες.

Κεφάλαιο 7

7. Μετρικά αξιολόγησης

7.1 Γενικά

Όταν πραγματοποιούμε μια ανάλυση πρέπει να θεσπίσουμε κάποια μετρικά που θα μας βοηθήσουν να αξιολογήσουμε την ανάλυσή μας. Όπως προαναφέραμε υπάρχουν 7 crashes σε nominal συνθήκες και στην σελίδα 49 κάνουμε μία ανάλυση για αυτά.

Θα θεσπίσουμε κάποιες έννοιες ώστε να μπορούμε να περιγράψουμε καλύτερα την ανάλυσή μας.

7.2 Crash Point

Θεωρούμε ως χρονικό σημείο crash το τελευταίο timestamp σε ένα πείραμα που γνωρίζουμε ότι είχε crash (σελίδα 23) πριν να δούμε timestamps που προδίδουν booting process(σελίδα 40). Αυτό το timestamp θα το ονομάσουμε **crash point**. Για παράδειγμα crash point είναι το εξής:

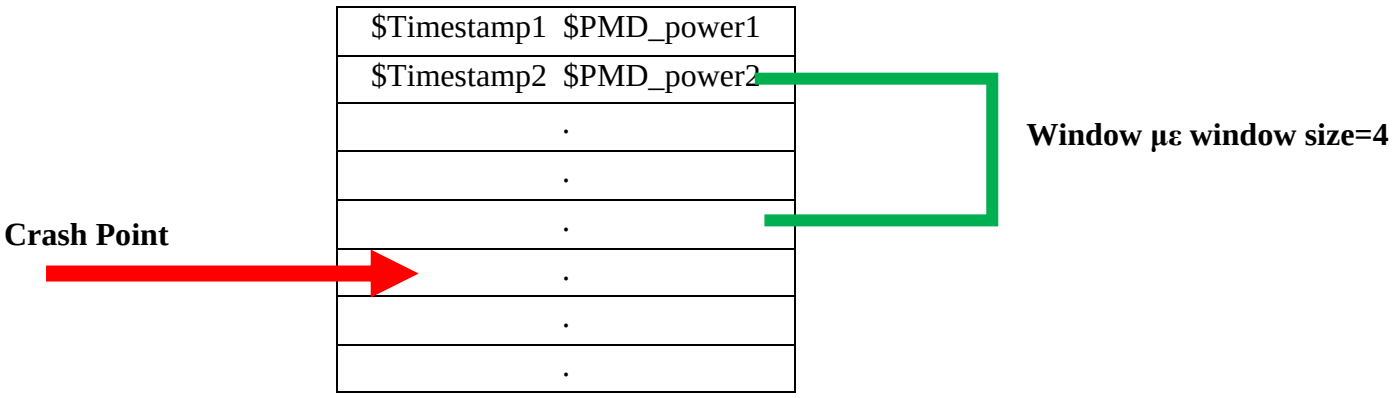
```
1533320634 11366 7265 2346 2342 69 83 65 45 43 44 41 980 937 1500 1500 78
1533320635 11398 7257 2384 2348 69 83 65 45 43 44 41 979 937 1500 1500 78
1533320637 11382 7148 2379 2322 68 83 65 45 43 44 41 979 938 1500 1500 78
1533320638 11272 7288 2367 2352 69 83 65 45 43 44 41 980 938 1500 1500 78
1533320639 11382 7050 2379 2360 69 83 65 45 43 44 41 979 938 1500 1500 78
1533320640 11296 7335 2388 2348 69 83 65 45 43 43 41 980 938 1500 1500 78
1435079383 4733 7452 2552 2817 62 75 62 42 39 41 39 979 942 1500 1499 78
1435079384 8320 8237 3741 3653 64 70 63 42 39 41 40 980 943 1500 1499 78
```

⇨ Το timestamp=1533320640 είναι crash point

7.3 Window

Θέλουμε να δούμε αν οι τιμές που προηγούνται ενός crash έχουν σημαντικό ρόλο.

Για κάθε χρονικό σημείο crash παίρνουμε X monitor vectors που προηγούνται του crash. **Θα πάρουμε το σύνολο αυτών των X monitor vectors και θα το ονομάσουμε window.** Το μέγεθος αυτού του window **(window size)** είναι ο αριθμός των monitor vectors του· επομένως window size=X.

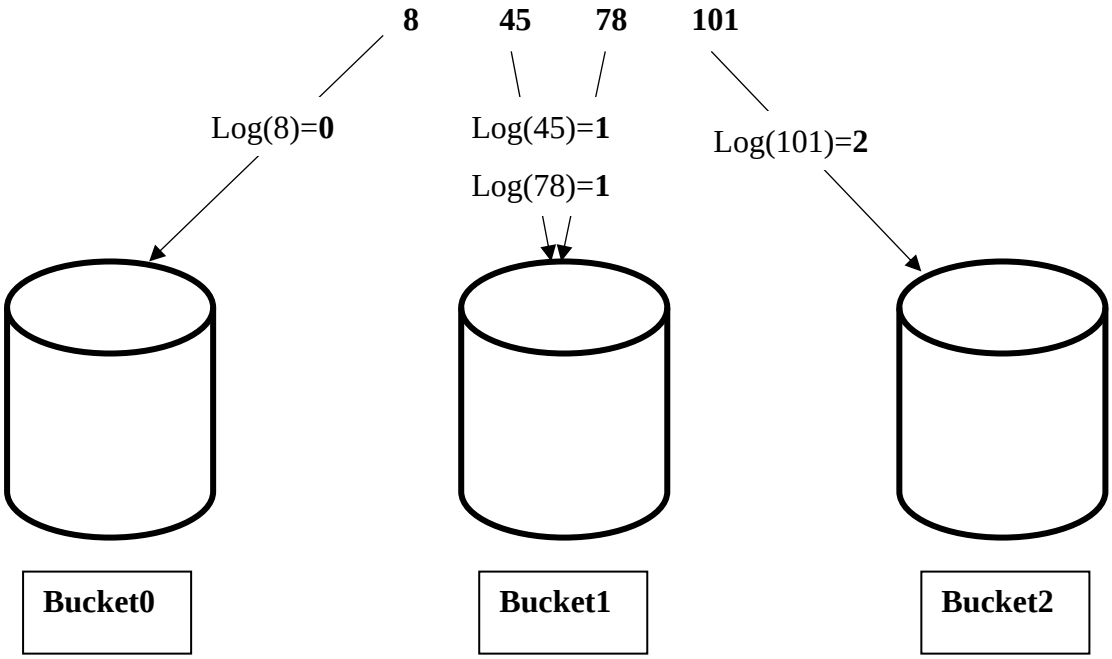


7.4 Buckets

Παίρνουμε όλες τις τιμές ενός μετρικού (έστω PMD power) που έχει το monitor file.Αφού υπάρχουν πολλές διαφορετικές τιμές, θέλουμε να κατατάξουμε τις τιμές σε "δοχεία" με κοινά χαρακτηριστικά. Θα ονομάσουμε αυτά τα δοχεία **buckets**. Κάθε φορά τα buckets είναι διαφορετικά, ανάλογα με το πως επιθυμούμε να τα δημιουργήσουμε (δηλαδή τι θέλουμε να έχουν κοινό).

Για παράδειγμα μπορούμε να δημιουργήσουμε buckets κάνοντας στις τιμές του μετρικού τον λογάριθμο με βάση 10. Έστω ότι έχουμε τις τιμές: 8, 45,78,101 και δημιουργούμε buckets με log10. Τότε έχουμε 3 buckets. Το bucket0 με μέγεθος 1 (εδώ ανήκουν όλες οι τιμές που δίνουν log10=0), το bucket1 με μέγεθος 2 και το bucket2 με μέγεθος 1.

Ο μέγιστος αριθμός buckets που μπορούμε να έχουμε σε ένα μετρικό, είναι όσες είναι και οι μοναδικές τιμές του μετρικού και αντίστροφα, ο ελάχιστος αριθμός buckets που μπορούμε να έχουμε είναι 1 (όλες οι τιμές ανήκουν σε 1 bucket).



7.5 Coverage of Distribution

Κάθε bucket έχει ένα μέγεθος. Αν προσθέσουμε το μέγεθος όλων των buckets, θα έχουμε το πλήθος των τιμών του συγκεκριμένου feature. Αν πάρουμε την αναλογία μεταξύ του μεγέθους ενός bucket και του μεγέθους όλων των buckets, τότε έχουμε το ποσοστό των τιμών του συγκεκριμένου feature που ανήκουν στο bucket συγκριτικά με όλα τα buckets.

Θα ονομάσουμε το ποσοστό αυτό **Coverage of Distribution.**

Ο μαθηματικός τύπος είναι ο εξής:

Coverage of Distribution for bucket0=sizeof(bucket_0) / Sum(sizeof(bucket_i))

Όπου i=0-num_of_buckets

Παράδειγμα:

Έχουμε 3 buckets:

- bucket1 με size=5 bucket2 με size=9 bucket3 με size=1
- size of all the buckets=15

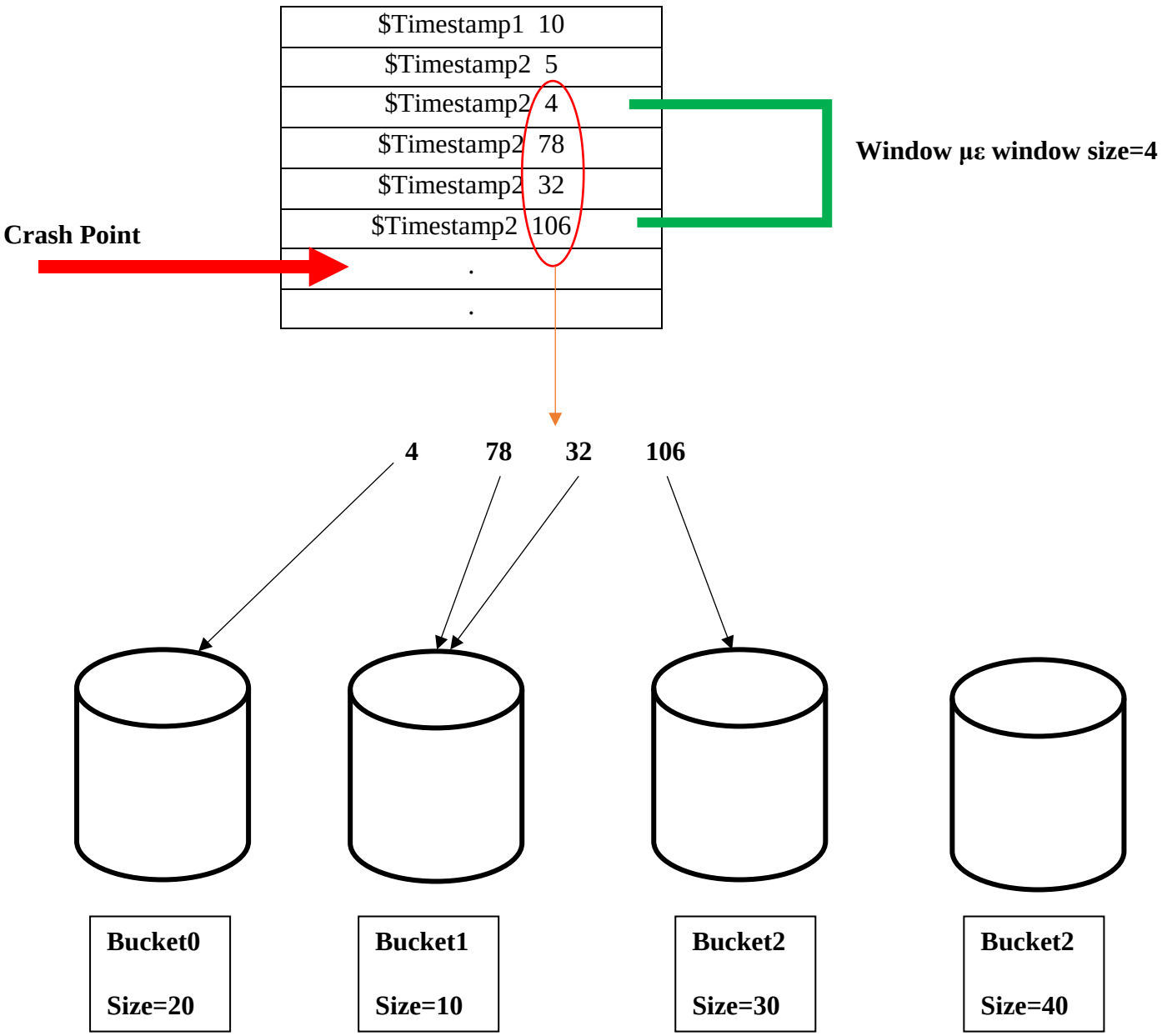
Επομένως έχουμε ότι:

Coverage of Distribution of:

- bucket1=5/15=0.33 bucket2=9/15=0.60 bucket3=1/15=0.067

7.6 Window Coverage of Distribution

Παίρνουμε όλες τις τιμές ενός feature σε ένα ή πολλαπλά window. Αντιστοιχούμε αυτές τις τιμές στα buckets τα οποία ανήκουν. Παίρνουμε από κάθε bucket το Coverage of Distribution του και τα αθροίζουμε. Το άθροισμα όλων των Coverage of Distribution των buckets που ανήκουν οι τιμές του window, θα το ονομάσουμε **Window Coverage of Distribution.**



Total size of buckets=100

Coverage of Distribution of:

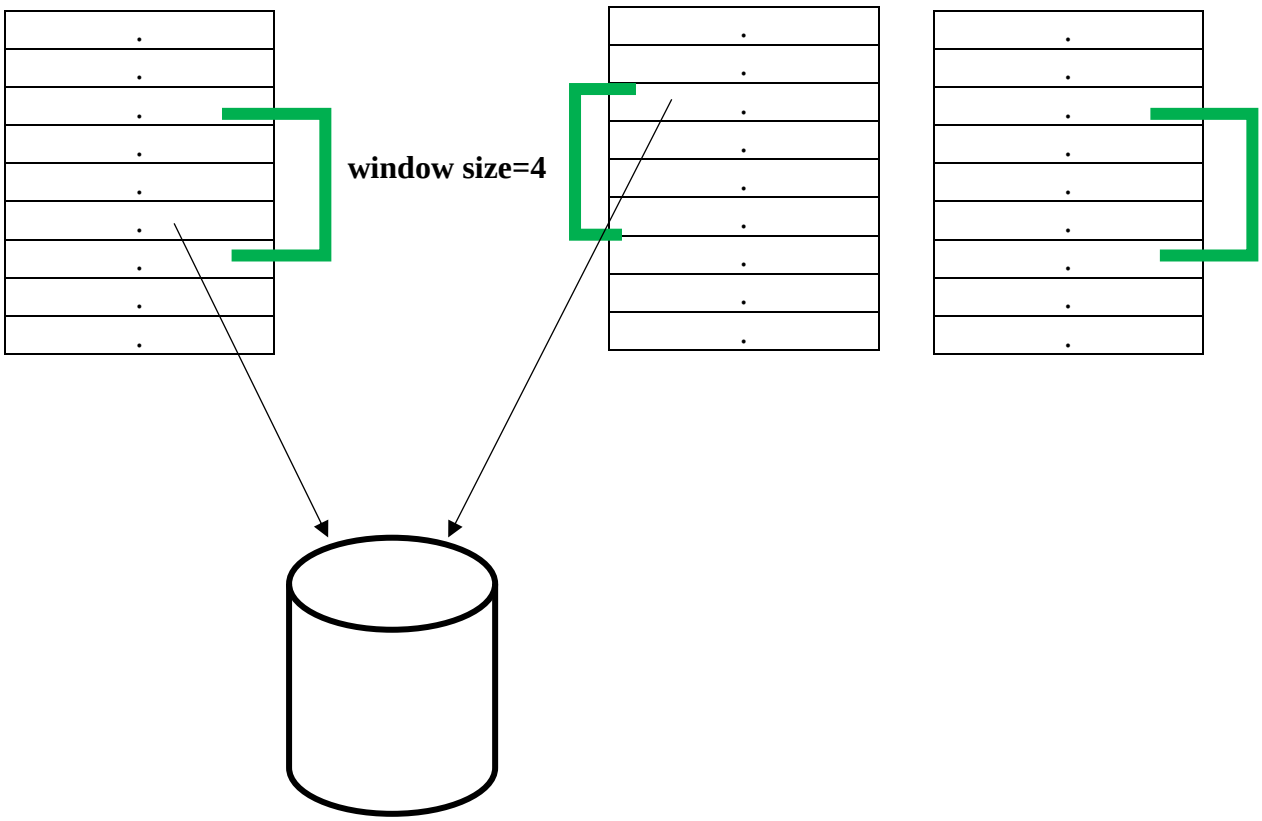
bucket0=0.20 bucket1=0.10 bucket2=0.3 bucket3=0.4

Window Coverage of Distribution = Coverage of Distribution of (bucket1+bucket2+bucket3) **= 0.6**

7.7 Coverage of Crashes

Όπως προαναφέραμε, έχουμε 7 καταγεγραμμένα πειράματα από crashes. Βλέπουμε για κάθε window των 7 crashes (window size για κάθε crash το ίδιο) αν περιέχει τιμές feature που αντιστοιχούν σε ένα συγκεκριμένο bucket. Για κάθε window που περιέχει τιμή που αντιστοιχεί σε αυτό το bucket, το προσθέτουμε στην λίστα που αυτό το bucket "καλύπτει". Το ποσοστό των crash windows που ένα bucket καλύπτει, θα το ονομάσουμε **Coverage of Crashes.**

Έστω ότι υπάρχουν 3 crashes(άρα και 3 windows).



Το bucket έχει τιμές από 2 windows

Επομένως, Coverage of Crashes = 2/3 =**0.66**

Κεφάλαιο 8

8. Crash Data Analysis

8.1 Γενικά

Σε αυτό το κεφάλαιο θα επιχειρήσουμε να κάνουμε μια αρχική απλοϊκή ανάλυση των δεδομένων που έχουμε συλλέξει μέσω του framework πριν από ένα crash point.Ο σκοπός αυτής της ανάλυσης είναι να αναλύσουμε τα δεδομένα πριν το crash point, προσπαθώντας να δούμε αν αυτά μπορούν να μας προσδώσουν ότι επέρχεται crash.

Ο λόγος που μπορούμε να πραγματοποιήσουμε αυτή την ανάλυση είναι λόγω της συνεισφοράς του framework. Επειδή το framework έπαιρνε δεδομένα ασταμάτητα, έχουμε τόσο δεδομένα πριν από ένα crash point, αλλά και πολλά δεδομένα για χρονικές στιγμές που δεν επερχόταν κάποιο crash. Να αναφέρουμε ότι τα 7 nominal crashes που παρουσιάστηκαν, δεν είναι πολλά για μια ανάλυση δεδομένων, κάτι που επηρεάζει την ανάλυση μας.

Για να υπήρξαν αυτά τα crashes σε nominal συνθήκες, πιστεύουμε ότι κάτι παράξενο/ιδιόμορφο έγινε στον υπολογιστή που προκάλεσε το crash. Τα features που μαζεύουμε από το monitor, ίσως είναι ικανά να μας δείξουν τι προκάλεσε το crash.

Με αυτή την ανάλυση και με μελλοντικές έρευνες πάνω σε αυτή θέλουμε να εξευρεθεί τρόπος να προβλέπουμε πότε επέρχεται crash. Προβλέποντας τα crashes θα έχει πολλά οφέλη για την επιστημονική κοινότητα. Θα παίρνονται μέτρα για να αντιμετωπίζονται τα crashes προσφέροντας:

- μεγαλύτερη αξιοπιστία στους χρήστες των υπολογιστών
- δεν θα χάνονται δεδομένα από crashes
- καλύτερη κατανόηση των λειτουργιών του υπολογιστή
- κ.α.

8.2 Τρόποι ανάλυσης

Προσπαθήσαμε να κάνουμε ανάλυση των δεδομένων πριν από ένα crash point (window) και να το συγκρίνουμε με τα υπόλοιπα δεδομένα που έχουμε συλλέξει με διαφορετικούς τρόπους. Ακολουθήσαμε την εξής διαδικασία:

1. Αρχικά ξεκινήσαμε παίρνοντας windows μεγέθους 10 από κάθε crash που είχαμε(δηλαδή συνολικά 70 monitor vectors). Αφαιρέσαμε τα 70 αυτά monitor vectors από το monitor και επομένως τώρα έχουμε 2 σύνολα monitor vectors· τα 70 monitor vectors (θα το ονομάσουμε **allWindows**) και το monitor που του έχουν αφαιρεθεί τα 70 monitor vectors (θα το ονομάσουμε **subMonitor**)
2. Για να δημιουργήσουμε τα buckets μας, θα χρησιμοποιήσουμε τον λογάριθμο βάσης e ($\log_e$—παίρνουμε μόνο το ακέραιο κομμάτι του λογαρίθμου).Δημιουργούμε buckets για όλες τις μετρήσεις, όλων των features του subMonitor.
3. Ακολούθως, για κάθε μία από τις 70 τιμές κάθε feature στο allWindows, υπολογίσαμε το bucket στο οποίο ανήκει, και ακολούθως το **Coverage of Distribution** του bucket. Ιδανικά θέλουμε οι τιμές να ανήκουν σε buckets με χαμηλό **Coverage of Distribution,** γιατί αυτό δείχνει ότι αυτές οι τιμές είναι σπάνιες. Αν κάποια τιμή ανήκει σε κάποιο bucket με **Coverage of**

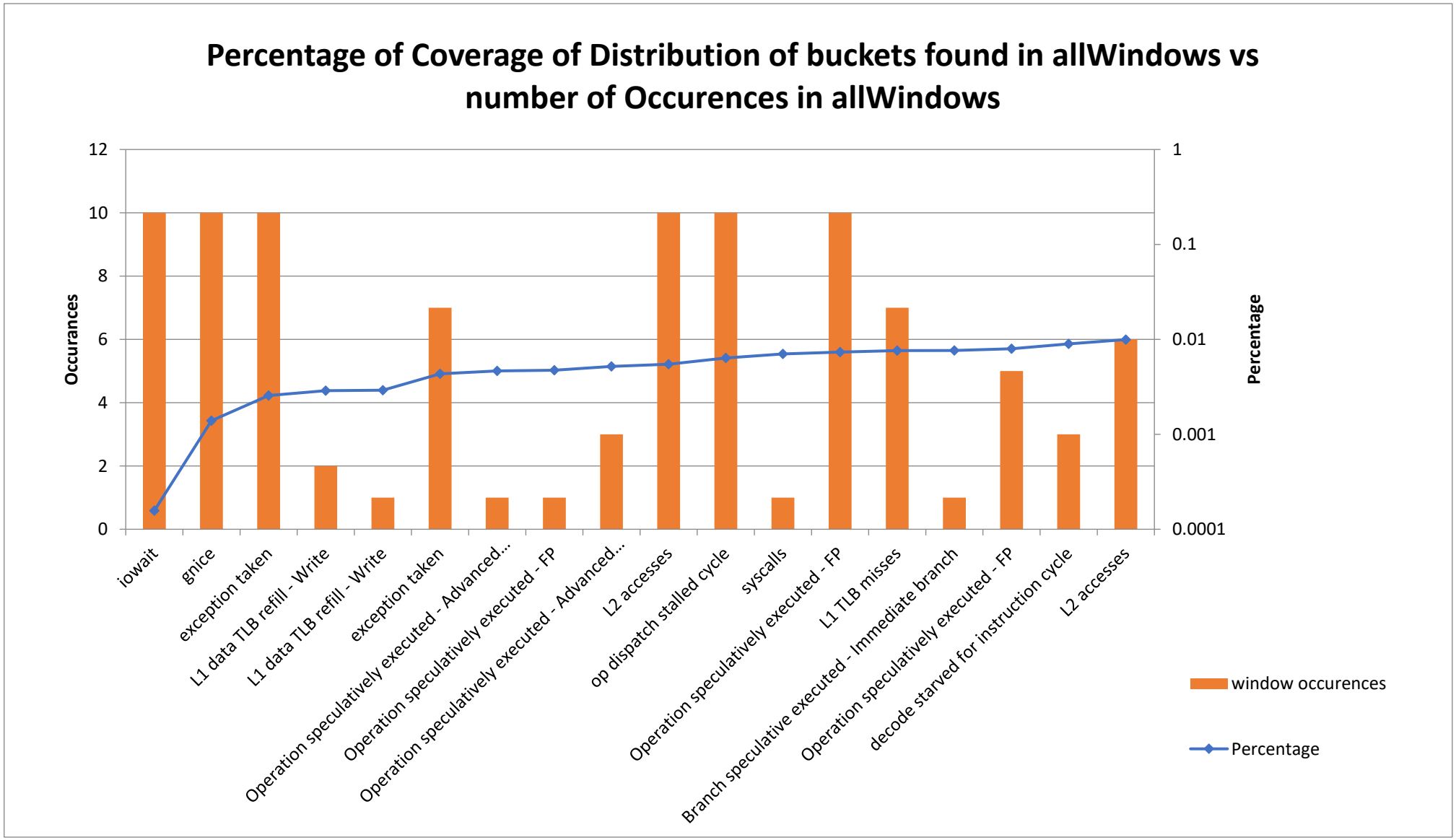
Distribution=0, τότε αυτό συνεπάγεται ότι αυτή η τιμή είναι μοναδική σε αυτό το bucket, κάτι που την κάνει πολύ σπάνια και επιθυμητή για την ανάλυσή μας. Παρόλα αυτά, δεν βρήκαμε τέτοιες τιμές.

4. Έπειτα, αφού βρήκαμε το **Coverage of Distribution** για κάθε τιμή, υπολογίζουμε το **Window Coverage of Distribution** για κάθε feature στο allWindows**.** Ιδανικά θέλουμε το **Window Coverage of Distribution** να είναι χαμηλό, γιατί αυτό δείχνει ότι όλες οι τιμές στο συγκεκριμένο feature είναι σπάνιες.
5. Τέλος, πήραμε τα buckets του **βήματος 3** και υπολογίσαμε για κάθε ένα το **Coverage of Crashes** του. Ιδανικά θέλουμε ψηλό **Coverage of Crashes,** γιατί αυτό δείχνει ότι το συγκεκριμένο bucket εμφανίζεται σε πολλές περιπτώσεις πριν από ένα crash, επομένως μπορεί να είναι αιτία, ή μέρος της αιτίας, που προκαλείται το crash.

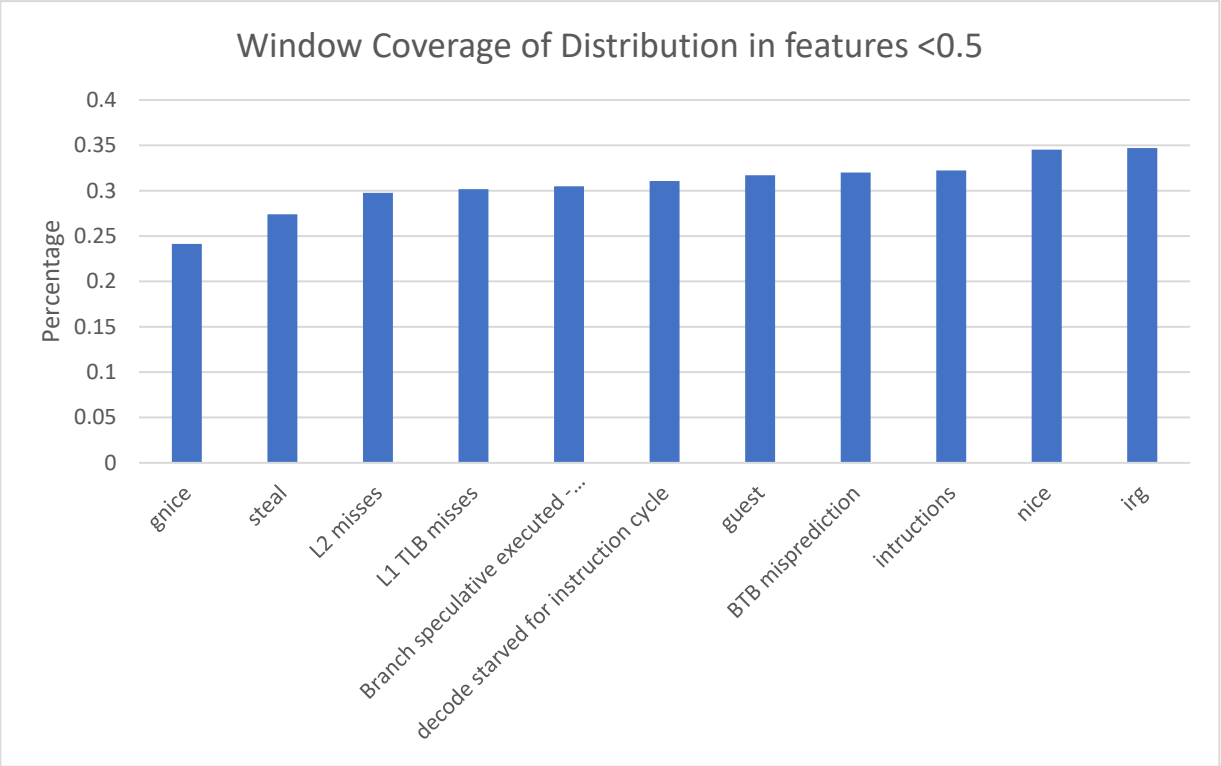
8.3 Αποτελέσματα

Αφού συλλέξαμε τα αποτελέσματα μας όπως περιγράψαμε πιο πάνω βρήκαμε τις τιμές που στο **βήμα 3** μας έδιναν το χαμηλότερο **Coverage of Distribution.** Πιο κάτω φαίνονται οι τιμές που έδωσαν **Coverage of Distribution < 0.01.**

Τις συγκρίνουμε με τον αριθμό των φορών που εμφανίστηκαν στο allWindows


Percentage of Coverage of Distribution of buckets found in allWindows vs number of Occurences in allWindows
Occurances
Percentage
12
10
8
6
4
2
0
1
0.1
0.01
0.001
0.0001
iowait
gnice
exception taken
L1 data TLB refill - Write
L1 data TLB refill - Write
exception taken
Operation speculatively executed - Advanced...
Operation speculatively executed - FP
Operation speculatively executed - Advanced...
L2 accesses
op dispatch stalled cycle
syscalls
Operation speculatively executed - FP
L1 TLB misses
Branch speculative executed - Immediate branch
Operation speculatively executed - FP
decode starved for instruction cycle
L2 accesses
window occurences
Percentage

Ακολούθως βρήκαμε τα features που στο **βήμα 4** μας έδιναν το χαμηλότερο **Window Coverage of Distribution (<0.5).**



Αυτή η μέθοδος δεν μας βοήθησε ιδιαίτερα στην ανάλυση μας λόγω των ψηλών του ποσοστών. Το **Window Coverage of Distribution** μας δίνει πληροφορίες για όλες τις τιμές ενός feature και όχι για μεμονωμένες τιμές.

Στο τέλος επικεντρωθήκαμε στις τιμές του **βήματος 3** με **Coverage of Distribution < 0.01**, ταξινομώντας τες σύμφωνα με το Coverage of Crashes.

8.3.4 Συμπεράσματα

Από τα αποτελέσματα του Coverage of Crashes έχουμε τα εξής:

- Κανένα bucket, δεν "καλύπτει" όλα τα windows.
- Ο μέγιστος αριθμός windows που ένα bucket καλύπτει είναι 4.
- Κανένα bucket του βήματος 3 δεν "καλύπτει" το window του πειράματος **SPEC2006-tonto**

Από τα πιο πάνω καταλήξαμε στα εξής συμπεράσματα:

- Χρειαζόμαστε περισσότερα από ένα bucket κάποιου feature για να προβλέψουμε crash αφού κανένα σπάνιο bucket (**Coverage of Distribution < 0.01**) δεν μας δίνει Coverage of Crashes=1.
- Αφού κανένα από τα buckets που πήραμε δεν "κάλυπτε" το πείραμα SPEC2006-tonto, οι τρόποι για να επιλυθεί αυτό είναι:
    - Να πάρουμε buckets με μεγαλύτερο **Coverage of Distribution (π.χ. Coverage of Distribution<0.1)**
    - Να επαναλάβουμε ολόκληρη την διαδικασία μεγαλώνοντας το window size (π.χ. 100 'η 1000)

Έχουμε ξεκινήσει την υλοποίηση και των δύο αυτών τρόπων, όμως δεν έχουμε ακόμη τελειώσει αυτό το κομμάτι της ανάλυσης. Θα είναι μέρος της μελλοντικής εργασίας που πρέπει να γίνει, ώστε να ολοκληρωθεί η μελέτη. Παρόλα αυτά, πρέπει να είμαστε προσεκτικοί στα εξής:

- Όσο παίρνουμε buckets με μεγαλύτερο Coverage of Distribution, τόσο μεγαλώνουμε το ποσοστό false positive για crash.
    - Για παράδειγμα, αν έχουμε αν έχουμε ένα bucket με Coverage of Distribution =0.5 αυτό σημαίνει ότι σε αυτό το bucket ανήκει το 50% των μετρήσεων αυτού του feature, επομένως 0.5*5M=2.5M . Αν χρησιμοποιήσουμε αυτό το bucket για να προβλέπουμε πότε επέρχεται crash, τότε αυτό σημαίνει ότι πολλές φορές θα προβλέπαμε λάθος (υπάρχουν 2.5M μετρήσεις για αυτό το bucket). Αυτές τις λάθος προβλέψεις για crash, τις ονομάζουμε **false positives** και επιθυμούμε να είναι όσο το δυνατό λιγότερες.
- Όσο μεγαλώνουμε το window size, πηγαίνουμε όλο και πιο μακριά από το crash point, επομένως τα features που θα νομίζουμε ότι έχουν σημασία στο crash, να μην έχουν, αφού θα είμαστε τόσο μακριά από το crash point.

Κεφάλαιο 9

9. Σχετική Εργασία

9.1 Prometheus - Grafana

Υλοποιήσαμε ένα framework ώστε να μπορούμε να παρακολουθούμε τη συμπεριφορά του συστήματος. Υπάρχουν και άλλα framework παρακολούθησης που χρησιμοποιούνται ευρέως με τα κυριότερα να είναι το **Prometheus** και το **Grafana** τα οποία είναι και τα δύο Open Source. Το Prometheus εργαλείο είναι υλοποιημένο στο framework του Grafana με επιπλέον προστιθέμενες λειτουργίες. Και τα δύο εργαλεία είναι εργαλεία monitoring ακριβώς όπως και το framework που υλοποιήσαμε. Αυτά τα framework, υπερέχουν στην διαδραστικότητα (έχουν GUI) και στα γραφήματα συγκριτικά με την δική μας υλοποίηση, όμως παρόλα αυτά κανένα από αυτά δεν θα μπορούσε να εξυπηρετήσει τον σκοπό για τον οποίο χρειαζόμασταν το δικό μας framework και για αυτό ήταν αναγκαία η δημιουργία του δικού μας framework.

Μέσω του UniServer Project δημιουργήθηκε η ανάγκη του framework μας, μιας και έπρεπε να λαμβάνουμε πολύ συγκεκριμένα μετρικά από τους Server X-Gene 2 και X-Gene 3. Τα μετρικά αυτά είναι πολύ συγκεκριμένα γιατί επιλέχτηκαν σύμφωνα με το επιστημονικό paper του University of Athens με το όνομα “Statistical Analysis of Multicore CPUs Operation in Scaled Voltage Conditions” όπως προαναφέραμε. Αυτά τα μετρικά λαμβάνονται από perf stat και i2cget εντολές. Τα δύο εργαλεία (Grafana και

Prometheus) **δεν μας δίνουν την δυνατότητα να πάρουμε perf stat και i2cget εντολές για τους X-Gene 2 και X-Gene 3** και άρα να πάρουμε τις μετρήσεις που θέλουμε. Μπορούν παρόλα αυτά να μας βοηθήσουν στο visualization.

Επομένως ήταν αδίρητη ανάγκη η δημιουργία του framework για την παρακολούθηση των συγκεκριμμένων Servers αφού τα εργαλεία Prometheus και Grafana δεν θα μας έδιναν ότι χρειαζόμασταν.

Κεφάλαιο 10

10. Ανασκόπηση

10.1 Συνεισφορά

Η Συνεισφορά αυτής της διπλωματικής εργασίας βασίζεται σε τρεις πυλώνες.

1. Ανάπτυξη του framework για παρακολούθηση της συμπεριφοράς του υπολογιστή.
2. Characterization του Polaris Application
3. Αρχική Data Crash Analysis

Το framework που αναπτύξαμε, έχει σκοπό να βοηθήσει την ακαδημαϊκή κοινότητα και τις εταιρίες Υπολογιστών, να συλλέγουν δεδομένα για την συμπεριφορά του υπολογιστή. Ακόμη, με την βοήθεια του framework, υπάρχει ένας συστηματικός τρόπος καταγραφής πειραμάτων/benchmarks που σε συνδυασμό με τα δεδομένα που καταγράφονται, μπορούν να εξαχθούν χρήσιμα συμπεράσματα τόσο για το πείραμα, όσο και για τον υπολογιστή. Αυτό, κάνει το framework ένα εργαλείο για καλύτερη και ευκολότερη έρευνα. Επιπλέον, με τα πολλά δεδομένα που συλλέγει το framework, μπορεί να παραχθούν πληροφορίες που δεν θα ήταν αλλιώς δυνατό.

Δείξαμε την εφαρμογή του framework στο characterization της εφαρμογής Polaris. Μέσω του UniServer Project, μας είχε ζητηθεί να κάνουμε characterization της Polaris εφαρμογής. Από το characterization, προτείναμε τον τρόπο που θα τρέχουμε την εφαρμογή στο UniServer Project, βρήκαμε την χαμηλότερη τάση που η εφαρμογή είναι ασφαλής να τρέξει και αναλύσαμε την εξοικονόμηση ενέργειας που θα πετύχουμε με την πιο χαμηλή τάση. Το framework μας βοήθησε στο characterization, καταγράφοντας τα πειράματα και παίρνοντας τις μετρήσεις της κατανάλωσης ενέργειας.

Επιπλέον, δείξαμε την εφαρμογή του framework και στο Data Crash Analysis. Λόγω των πολλών δεδομένων που λάμβανε το framework συνεχώς, είχαμε δεδομένα για στιγμές όπου παρουσιαζόταν κάποιο crash. Με τα δεδομένα, προσπαθήσαμε να κάνουμε μία αρχική ανάλυση για να μπορούμε να προβλέψουμε crash. Η ανάλυσή μας δεν μας ανέδειξε κάποιο συστηματικό τρόπο για ανίχνευση επερχόμενου crash. Όμως, η ανάλυση μας δεν έχει ολοκληρωθεί και θέλουμε να την ολοκληρώσουμε σε μελλοντική εργασία. Τα πιθανά οφέλη αν εξευρεθεί τρόπος πρόβλεψης ενός crash είναι πολλά.

10.2 Μελλοντική εργασία

Η παρούσα Διπλωματική εργασία αποτελεί την αρχή του Data Crash Analysis που αναφέραμε στην σελίδα 49 αφού δεν έχει ακόμη ολοκληρωθεί. Αναφερθήκαμε στο τέλος της ανάλυσης πως μπορεί η ανάλυση να γίνει πιο πλήρης. Χρειάζεται να συνεχίσουμε την ανάλυσή μας, εξετάζοντας

- μεγαλύτερα windows
- συνδυασμούς μεταξύ buckets
- και έχοντας περισσότερα crash points(αυτή την στιγμή έχουμε μόνο 7) που περιορίζει την ανάλυσή μας

Θα προσπαθήσουμε να συνεχίσουμε και να ολοκληρώσουμε την ανάλυση που μπορεί να μας προσδώσει τον τρόπο για να προβλέπουμε πότε επέρχεται crash. Η πρόβλεψη ενός crash έχει πολλά να προσφέρει στην επιστημονική κοινότητα και στον ιδιωτικό τομέα. Πιστεύω ότι χρειαζόμαστε περισσότερα δεδομένα και από πιο πολλές πηγές. Με περισσότερα δεδομένα, και μια πιο σύνθετη ανάλυση, χρησιμοποιώντας ίσως και αλγορίθμους μηχανικής μάθησης, ίσως υπάρχει η δυνατότητα για πρόβλεψη ενός crash.

Βιβλιογραφία

[1] CAL 2018 – "Statistical Analysis of Multicore CPUs Operation in Scaled Voltage Conditions", M.Kaliorakis, A.Chatzidimitriou, G.Papadimitriou, D.Gizopoulos, IEEE Computer Architecture Letters, vol. 17, no. 2, pp. 109-112, July-December 2018

[2] Dr. Biehl – Lecture Notes in Intelligent Systems course at the University of Groningen, 2017.

[3] Dr. Biehl – Lecture Notes in Neural Networks course at the University of Groningen, 2017.

[4] UniServer Consortium Partners – “D4.1 Hardware Exposure Interface (HEI) and Error Handlers Specification", 2016.

[4] UniServer Consortium Partners – "D7.1 First Report on Metrics of Success Against State-of-the-Art ", 2017.

[5] Prometheus - Monitoring system & time series database, https://prometheus.io/

[6] Grafana - The open platform for analytics and monitoring, https://grafana.com/

Παράρτημα Α

Κώδικας **Perf_counters.sh**

```
#!/bin/bash
dmidecode>>/home/uniserver_measure/booting_log
sudo cpupower frequency-set -f 2400M
printf "\n#######################\n">>/home/uniserver_measure/booting_log
saveFolder="/home/uniserver_measure"
fileName="monitor"
while :
do
	perf stat -a -o deleteme -e instructions,cycles,r16,r17,l3c0/read-miss/,branch-misses,raw_syscalls:sys_enter,r107,r9,r1,r5,r108,r109,r102,r7a,r78,r4d,r73,r74,r75 sleep 1

	printf "\n%s " "$(date +%s)">>$saveFolder/$fileName

	#PMD, SoCVRD(Uncore), DIMM1, DIMM2 power
	PMD=`i2cget -y 1 0x2f 0x20 w`
	PMD=$((PMD *1000))
	PMD_mil=`i2cget -y 1 0x2f 0x21 w`
	PMD_mil=$((PMD + PMD_mil))

	SoC=`i2cget -y 1 0x2f 0x22 w`
	SoC=$((SoC *1000))
	SoC_mil=`i2cget -y 1 0x2f 0x23 w`
	SoC_mil=$((SoC + SoC_mil))

	DIM1=`i2cget -y 1 0x2f 0x24 w`
	DIM1=$((DIM1 *1000))
	DIM1_mil=`i2cget -y 1 0x2f 0x25 w`
	DIM1_mil=$((DIM1 + DIM1_mil))

	DIM2=`i2cget -y 1 0x2f 0x26 w`
	DIM2=$((DIM2 *1000))
	DIM2_mil=`i2cget -y 1 0x2f 0x27 w`
	DIM2_mil=$((DIM2 + DIM2_mil))
```

```
printf "%d %d %d %d " $PMD_mil $SoC_mil $DIM1_mil $DIM2_mil >>$saveFolder/$fileName


#PMD, SoC, SoCVRD(Uncore), DIMM0, DIMM1, DIMM2, DIMM3 temp

printf "%d %d %d %d %d %d %d " $(i2cget -y 1 0x2f 0x13 w) $(i2cget -y 1 0x2f 0x10 w) $(i2cget -y 1 0x2f 0x11 w) $(i2cget -y 1 0x2f 0x14 w) $(i2cget -y 1 0x2f 0x15 w) $(i2cget -y 1 0x2f 0x16 w) $(i2cget -y 1 0x2f 0x17 w)>>$saveFolder/$fileName


#PMD, SoCVRD(Uncore), DIMM1, DIMM2 voltage

printf "%d %d %d %d " $(i2cget -y 1 0x2f 0x34 w) $(i2cget -y 1 0x2f 0x35 w) $(i2cget -y 1 0x2f 0x36 w) $(i2cget -y 1 0x2f 0x37 w)>>$saveFolder/$fileName


#DIMM refresh Rate,DIMM frequency,L3 frequency

printf "%d %d %d " $(i2cget -y 1 0x2f 0x40 w) $(i2cget -y 1 0x2f 0x41 w) $(i2cget -y 1 0x2f 0x42 w)>>$saveFolder/$fileName


#utilization usr, nice, sys, iowait, irg, soft, steal, guest, gnice

mpstat 1 1 | grep Average | awk '{printf "%.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f ", $4, $5, $6, $7, $8, $9, $10, $11, $12}'>>$saveFolder/$fileName


#perf stats intructions, cycles, L2 accesses, L2 misses, L3 misses, branch misses, syscalls

sed -i 's/,//g' deleteme

cat deleteme | grep instructions | awk '{printf $1 " "}'>>$saveFolder/$fileName

cat deleteme | grep cycles | awk '{printf $1 " "}'>>$saveFolder/$fileName

cat deleteme | grep r16 | awk '{printf $1 " "}'>>$saveFolder/$fileName

cat deleteme | grep r17 | awk '{printf $1 " "}'>>$saveFolder/$fileName

cat deleteme | grep read-miss | awk '{printf $1 " "}'>>$saveFolder/$fileName

cat deleteme | grep branch | awk '{printf $1 " "}'>>$saveFolder/$fileName

cat deleteme | grep raw_syscalls | awk '{printf $1 " "}'>>$saveFolder/$fileName


#perf stats L2 prefetch, exception taken, L1 misses, L1 TLB misses, decode starved for instruction cycle, op dispatch stalled cycle

cat deleteme | grep r107 | awk '{printf $1 " "}'>>$saveFolder/$fileName

cat deleteme | grep r9 | awk '{printf $1 " "}'>>$saveFolder/$fileName

cat deleteme | grep ' r1 ' | awk '{printf $1 " "}'>>$saveFolder/$fileName

cat deleteme | grep r5 | awk '{printf $1 " "}'>>$saveFolder/$fileName

cat deleteme | grep r108 | awk '{printf $1 " "}'>>$saveFolder/$fileName

cat deleteme | grep r109 | awk '{printf $1 " "}'>>$saveFolder/$fileName
```

#perf stats BTB misprediction, Branch speculative executed - Indirect branch, Branch speculative executed - Immediate branch, L1 data TLB refill - Write, Operation speculatively executed - Integer data processing, Operation speculatively executed - Advanced SIMD, Operation speculatively executed - FP

```
cat deleteme | grep r102 | awk '{printf $1 " "}'>>$saveFolder/$fileName
cat deleteme | grep r7a | awk '{printf $1 " "}'>>$saveFolder/$fileName
cat deleteme | grep r78 | awk '{printf $1 " "}'>>$saveFolder/$fileName
cat deleteme | grep r4d | awk '{printf $1 " "}'>>$saveFolder/$fileName
cat deleteme | grep r73 | awk '{printf $1 " "}'>>$saveFolder/$fileName
cat deleteme | grep r74 | awk '{printf $1 " "}'>>$saveFolder/$fileName
cat deleteme | grep r75 | awk '{printf $1 " "}'>>$saveFolder/$fileName

#frequency of cores - 11/02		- 11/13 corrected
cat /proc/cpuinfo | grep MHz | awk '/MHz/{i++}i==1{printf $4 " "}'>>$saveFolder/$fileName
cat /proc/cpuinfo | grep MHz | awk '/MHz/{i++}i==2{printf $4 " "}'>>$saveFolder/$fileName
cat /proc/cpuinfo | grep MHz | awk '/MHz/{i++}i==3{printf $4 " "}'>>$saveFolder/$fileName
cat /proc/cpuinfo | grep MHz | awk '/MHz/{i++}i==4{printf $4 " "}'>>$saveFolder/$fileName
cat /proc/cpuinfo | grep MHz | awk '/MHz/{i++}i==5{printf $4 " "}'>>$saveFolder/$fileName
cat /proc/cpuinfo | grep MHz | awk '/MHz/{i++}i==6{printf $4 " "}'>>$saveFolder/$fileName
cat /proc/cpuinfo | grep MHz | awk '/MHz/{i++}i==7{printf $4 " "}'>>$saveFolder/$fileName
cat /proc/cpuinfo | grep MHz | awk '/MHz/{i++}i==8{printf $4 " "}'>>$saveFolder/$fileName
done
```

Παράρτημα Β

Κώδικας**Script_Wrapper.sh**

```
#!/bin/bash
start=`date +%s`
execBinary=$1
saveFolder=/home/uniserver_measure/submitted_scripts
cp $execBinary $saveFolder/$start
param=$@
args=${param/\n/' '}
printf "#CorE_WorklOaD">>$saveFolder/$start
printf " %s" $args>>$saveFolder/$start
printf "\n">>$saveFolder/$start
for((j=0;j<1;j++))
do
        $@
done
cd $saveFolder
end=`date +%s`
mv $saveFolder/$start $saveFolder/$start'_'$end
```