



UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Software Policies and Techniques for Improving the Energy and Cost Efficiency of Cloud and High-Performance Computing Infrastructures

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Christos Kalogirou

This work is licensed under CC BY 4.0

April 2024



UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Software Policies and Techniques for Improving the Energy and Cost Efficiency of Cloud and High-Performance Computing Infrastructures

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Christos Kalogirou

This work is licensed under CC BY 4.0

April 2024



ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Πολιτικές και Τεχνικές Λογισμικού για τη Βελτίωση της Ενεργειακής και Οικονομικής Αποδοτικότητας Υποδομών Νέφους και Υπολογιστικής Υψηλής Επίδοσης

Διατριβή η οποία υποβλήθηκε για τη μερική εκπλήρωση
των υποχρεώσεων απόκτησης του Διδακτορικού Διπλώματος

Χρήστος Καλογήρου

This work is licensed under CC BY 4.0

Απρίλιος 2024



UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Software Policies and Techniques for Improving the Energy and Cost Efficiency of Cloud and High-Performance Computing Infrastructures

PhD Dissertation

Christos Kalogirou

Advisory committee

Christos/Χρήστος Antonopoulos/Αντωνόπουλος, Professor, University of Thessaly (Supervisor)

Nikolaos/Νικόλαος Mpellas/Μπέλλας, Professor, University of Thessaly

Spyros/Σπύρος Lalis/Λάλης, Professor, University of Thessaly

Examination committee

Christos/Χρήστος Antonopoulos/Αντωνόπουλος, Professor, University of Thessaly (Supervisor)

Nikolaos/Νικόλαος Mpellas/Μπέλλας, Professor, University of Thessaly

Spyros/Σπύρος Lalis/Λάλης, Professor, University of Thessaly

Georgios/Γεώργιος Gkoumas/Γκούμας, Associate Professor, National Technical University of Athens

Vana/Βάνα Kalogeraki/Καλογεράκη, Professor, Athens University of Economics and Business

Kostas/Κώστας Magoutis/Μαγκούτης, Associate Professor, University of Crete

Panagiota/Παναγιώτα Tsompanopoulou/Τσομπανοπούλου, Associate Professor, University of Thessaly

April 2024

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this PH.D. dissertation, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

The declarant

Christos Kalogirou

Acknowledgements

This dissertation is the result of research work conducted while I was pursuing my PhD degree in the Department of Electrical and Computing Engineering of University of Thessaly in Greece.

First and foremost, I would like to thank my mentors, Professors Christos D. Antonopoulos, Nikolaos Bellas, and Spyros Lalas from University of Thessaly. They have been supporting me and guiding me throughout my PhD studies to accomplish my goals. Their guidance and mentoring had a significant contribution on completing this work.

I would also like to thank my parents, Dimitra and Konstantinos, for believing in me and teaching me how to chase my dreams. Also, a special thanks to my friends because they were always supporting me, and always helping me to refill my energy and continue my work.

Moreover, I want to thank my friends and colleagues Panos Koutsovasilis and Manolis Maroudas with whom I shared my research career, and our discussions were very refreshing and educating.

Finally, I acknowledge the funding agencies which made this research possible with their financial support. These include the European Commission through the UniServer H2020 and SCoRPiO FP7 projects, and the European Social Fund and Greek national resources through the Centaurus project (Aristeia II action).

PhD Dissertation

Software Policies and Techniques for Improving the Energy and Cost Efficiency of Cloud and High-Performance Computing Infrastructures

Christos Kalogirou

Abstract

Energy consumption is a critical concern in the areas of Cloud- and High-Performance Computing (HPC) due to economic and environmental reasons. To fulfill the requirements of the compute jobs they serve, modern datacentres are equipped with multi-core processors and high amounts of Dynamic Random Access Memory (DRAM), which significantly increase the energy footprint of the nodes. Therefore, the minimization of energy consumption is a challenging undertaking, as infrastructure providers need, at the same time, to meet the quality of service (QoS) requirements of the running services and applications. Cloud infrastructure providers are contractually bound with Service Level Agreements (SLAs) with their users, and the violation of SLAs translates to penalties and thus cost. There are several mechanisms/techniques that can reduce the energy consumption of the datacentres, while also considering the QoS. VM consolidation targets to pack as many VMs as possible on the nodes, ideally without performance loss, in order to reduce the number of active nodes. Heterogeneity can also improve the energy efficiency of the nodes; accelerators such as GPUs are superior in terms of performance per Watt compared with the conventional CPUs. If heterogeneity is combined with approximate computing the energy gains can be even higher, as the relaxation of some calculations may introduce more energy gains and favor certain architectures. On the hardware side, manufacturers have introduced Voltage and Frequency Scaling (VFS), enabling the processor to operate at more power-efficient operating points, whenever there is an opportunity to do so without penalizing the performance. However, at the same time, CPU and DRAM manufacturers introduce guardbands to the operating parameters of the processors (supply voltage for each frequency step) and the DRAMs (supply voltage and refresh rate) to guarantee correct operation under adverse combinations of operating conditions. Those guardbands are unnecessarily wide for real-world execution scenarios and translate to increased power consumption.

In this Thesis, we focus on improving the energy efficiency of cloud infrastructures by reducing hardware guardbands. We formally model the problem of dealing with the trade-off between energy cost and potential SLA violations by considering the VM requirements and the fact that they change over time. Moreover, we introduce scheduling policies that use these models and exploit reduced margins for the CPU and the DRAMs to lower the energy cost of cloud datacentres. Finally, we combine heterogeneity with approximate computing in HPC applications, and we evaluate the trade-off between the quality of output and the energy footprint of these applications. More specifically:

We model the cost for cloud/edge infrastructure providers studying the effects of the conventional technique of VFS and the more aggressive approaches of voltage (undervolting) and frequency (overclocking) overscaling of the processor. We introduce models for estimating the failure rate for each of the aforementioned techniques (VFS, undervolting, overclocking). Our models capture the expected energy consumption of the tasks as well as potential penalties due to SLA violations. Our study shows up to 13.31% cost reduction in the case of undervolting compared with the operation of the processor at nominal settings.

As a next step, we use a simulated environment to investigate at-scale the exploitation of reduced voltage guardbands on the processor by cloud infrastructure providers. We introduce a VM scheduling and node configuration policy that exploits operation at reduced voltage margins for the processor, on top of the combination of the conventional techniques of VFS and VM consolidation. The policy is based on an analytic and architecture-agnostic model that captures the trade-off between energy cost and potential SLA violation penalties. We evaluate our approach and compare it against a state-of-the-art scheduling and configuration policy. We use node profiles derived from the characterization of commercially available server-class systems. Moreover, for our experiments we use realistic values for the energy price and the price of VMs. We provide results with both a constant and a dynamically varying energy price. Our results show gains up to 17.24% and 17.72% for the cost and the energy, respectively, compared with a state-of-art policy that combines VFS with VM consolidation.

We also explore the trade-off between energy efficiency and node overloads, to manage VM consolidation in a cost-aware manner. More specifically, we introduce a model that uses runtime information to control the load of the nodes, identifying and considering correlated behavior among colocated workloads. As the requirements of VMs are not known in advance, it is critical to consider recent VM behavior to adapt the target load for each node.

We combine this approach with reduced voltage margins for the processor and VFS. We show that our policy closely matches, or even outperforms two state-of-the-art policies that combine VM consolidation with VFS – the second one, also operating the CPU at reduced voltage margins. A key differentiation with prior policies is that our method does not require an unrealistic, in-advance analysis of the characteristics of the workload.

We also study the exploitation of reduced margins for the operating parameters of DRAMs, namely the supply voltage and refresh rate. In contrast with CPUs, where an error mainly translates to a system crash, for DRAMs we study the effect of potential penalties due to correctable single- and detectable double-bit errors. In analogy with the work on CPUs, we introduce an architecture-agnostic model and a novel workload management and system configuration policy. Our model captures all the aspects of VM management and DRAM configuration. The policy we introduce uses this model to exploit reduced margins of the operating parameters of DRAMs, combined with VM consolidation. As future systems will be equipped with high DRAM capacities, we study this problem for different memory sizes. We, again, evaluate our approach at-scale on a simulated environment, using realistic node parameters and realistic values for the energy price and VM pricing. Our results show that 29.53% and 34.84% cost and energy gains can be achieved, compared with a state-of-art policy that exploits VM consolidation. The gains are amplified as we move to nodes with a higher memory capacity.

Finally, we examine the effect of heterogeneous and approximate computing on the energy footprint of HPC applications. Result quality is not equally affected by all computations of a program; therefore we can perform selected computations approximately to improve the energy efficiency without heavily penalizing the output quality. In our work, we use 4 applications from diverse domains, ranging from real-world applications to kernels. We modify these applications to exploit both heterogeneity and approximate computing. We evaluate our research on heterogeneous systems (equipped with CPUs and GPUs), and we quantify the benefits of heterogeneity and approximate computing, considering the trade-off between energy consumption and the quality of the output. Our results show that heterogeneity and approximate computing can offer significant energy gains.

Λέξεις Κλειδιά

Cloud Computing; High-Performance Computing; Energy Efficiency; Cost Minimization;

Dynamic Workload; Energy-aware Cloud Policies; Heterogeneity; Approximate Computing

Διδακτορική Διατριβή

Πολιτικές και Τεχνικές Λογισμικού για τη Βελτίωση της Ενεργειακής και Οικονομικής Αποδοτικότητας Υποδομών Νέφους και Υπολογιστικής Υψηλής Επίδοσης

Χρήστος Καλογήρου

Περίληψη

Η κατανάλωση ενέργειας αποτελεί ένα σημαντικό πρόβλημα στον τομέα της υπολογιστικής νέφους και υπολογιστικής υψηλών επιδόσεων λόγω οικονομικών και περιβαλλοντικών αιτιών. Για να εξυπηρετήσουν τις απαιτήσεις των υπολογιστικών εργασιών, τα κέντρα δεδομένων είναι εξοπλισμένα με πολυπύρηνους επεξεργαστές και μεγάλης χωρητικότητας μνήμης τυχαίας προσπέλασης, που αυξάνουν σημαντικά το ενεργειακό αποτύπωμα των υπολογιστικών κόμβων. Επομένως, η ελαχιστοποίηση της κατανάλωσης ενέργειας είναι ένα δύσκολο εγχείρημα, καθώς οι πάροχοι υπολογιστικών υποδομών χρειάζονται την ίδια στιγμή να παρέχουν την απαιτούμενη ποιότητα για τις τρέχουσες υπηρεσίες και εφαρμογές. Οι πάροχοι υποδομών υπολογιστικής νέφους είναι δεσμευμένοι με συμφωνητικά επιπέδου υπηρεσιών με τους χρήστες τους και η παραβίαση αυτών των συμφωνητικών μεταφράζεται σε κυρώσεις για τον πάροχο που αυξάνουν το λειτουργικό κόστος. Υπάρχουν διάφοροι μηχανισμοί/τεχνικές που μπορούν να μειώσουν την κατανάλωση ενέργειας των κέντρων δεδομένων, ικανοποιώντας ταυτόχρονα την απαιτούμενη ποιότητα υπηρεσιών. Η ανακατανομή των εικονικών μηχανών στοχεύει στην τοποθέτηση όσο το δυνατόν περισσότερων εικονικών μηχανών στους κόμβους, ιδανικά χωρίς απώλεια επίδοσης, ώστε να μειωθεί ο αριθμός των ενεργών υπολογιστικών κόμβων. Επίσης, η ετερογένεια μπορεί να βελτιώσει την ενεργειακή αποδοτικότητα των υπολογιστικών κόμβων καθώς οι επιταχυντές όπως οι κάρτες γραφικών υπερέρχουν σε απόδοση ανά Watt σε σχέση με τους συμβατικούς επεξεργαστές. Αν η ετερογένεια συνδυαστεί με την προσεγγιστική υπολογιστική, τα ενεργειακά κέρδη μπορεί να είναι ευρύτερα, καθώς η χαλάρωση της ποιότητας κάποιων υπολογισμών μπορεί να ευνοήσει την ενεργειακή αποδοτικότητα, ειδικά σε ορισμένες υπολογιστικές αρχιτεκτονικές. Από την πλευρά του υλικού, οι κατασκευαστές έχουν εισάγει τη δυναμική κλιμάκωση τάσης και συχνότητας του επεξεργαστή, επιτρέποντάς τον να λειτουργεί σε πιο ενεργειακά αποδοτικά επίπεδα,

όταν τα χαρακτηριστικά του υπολογιστικού φορτίου προσφέρουν αυτό το παράθυρο ευκαιρίας, χωρίς να μειώνεται η απόδοση. Ταυτόχρονα όμως, οι κατασκευαστές επεξεργαστών και μνημών τυχαίας προσπέλασης εισάγουν περιθώρια ασφαλείας στις παραμέτρους λειτουργίας των επεξεργαστών (τάση για κάθε επίπεδο συχνότητας) και των μνημών (τάση και ρυθμός ανανέωσης) για να εγγυηθούν τη σωστή λειτουργία σε δυσμενείς συνθήκες. Αυτά τα όρια ασφαλείας είναι αρκετά απαισιόδοξα για τη λειτουργία σε κοινές, ρεαλιστικές συνθήκες, και αυξάνουν αχρείαστα την κατανάλωση ενέργειας.

Σε αυτή τη διατριβή, επικεντρωνόμαστε στη βελτιστοποίηση της ενεργειακής αποδοτικότητας για τους παρόχους υποδομών υπολογιστικής νέφους, χαλαρώνοντας τα περιθώρια ασφαλείας στη λειτουργία του επεξεργαστή και της μνήμης. Μοντελοποιούμε το πρόβλημα για να αξιολογήσουμε τη σχέση μείωσης του ενεργειακού κόστους και του ρίσκου πιθανών παραβιάσεων των συμφωνητικών επιπέδου υπηρεσιών, λαμβάνοντας υπόψιν τις υπολογιστικές απαιτήσεις των εικονικών μηχανών και το γεγονός ότι αυτές αλλάζουν με την πάροδο του χρόνου. Επιπλέον, εισάγουμε πολιτικές δρομολόγησης που χρησιμοποιούν αυτά τα μοντέλα και αξιοποιούν την τεχνική των μειωμένων περιθωρίων ασφαλείας των παραμέτρων λειτουργίας του επεξεργαστή και της μνήμης ώστε να μειώσουμε το ενεργειακό κόστος για τα κέντρα υπολογιστικής νέφους. Τέλος, συνδυάζουμε την ετερογένεια με την προσεγγιστική υπολογιστική για εφαρμογές υψηλών επιδόσεων και αξιολογούμε την επιλογή μεταξύ της διαχείρισης της ποιότητας του τελικού αποτελέσματος και της μείωσης του ενεργειακού αποτυπώματος αυτών των εφαρμογών. Πιο συγκεκριμένα:

Μοντελοποιούμε το κόστος για τους παρόχους υποδομών υπολογιστικής νέφους μελετώντας την επίδραση συμβατικών τεχνικών όπως της δυναμικής κλιμάκωσης της συχνότητας και της τάσης του επεξεργαστή καθώς και πιο επιθετικών προσεγγίσεων όπως της μείωσης της τάσης και του υπερχρονισμού. Παρουσιάζουμε μοντέλα που εκτιμούν την πιθανότητα αστοχίας (crash) των υπολογιστικών συστημάτων για τις παραπάνω τεχνικές. Τα μοντέλα μας λαμβάνουν υπόψιν την εκτιμώμενη κατανάλωση ενέργειας των εργασιών, όπως και τις ενδεχόμενες κυρώσεις λόγω της παραβίασης των συμφωνητικών επιπέδου υπηρεσιών. Η μελέτη μας δείχνει μείωση του κόστους λειτουργίας ως και 13.31% στην περίπτωση της μείωσης της τάσης σε σχέση με την κανονική λειτουργία του επεξεργαστή.

Σαν επόμενο βήμα, χρησιμοποιούμε ένα περιβάλλον προσομοίωσης για να μελετήσουμε σε κλίμακα την αξιοποίηση των μειωμένων ορίων ασφαλείας για τον επεξεργαστή για παρόχους υποδομών υπολογιστικής νέφους. Εισάγουμε μια πολιτική δρομολόγησης εικονικών

μηχανών και διαχείρισης των υπολογιστικών κόμβων που εκμεταλλεύεται τα μειωμένα όρια ασφαλείας λειτουργίας του επεξεργαστή, σε συνδυασμό με τις συμβατικές τεχνικές της δυναμικής κλιμάκωσης της τάσης και της συχνότητας του επεξεργαστή και της ανακατανομής των εικονικών μηχανών. Η πολιτική βασίζεται σε αναλυτικά μοντέλα που εφαρμόζονται ανεξαρτήτως της αρχιτεκτονικής και λαμβάνουν υπόψη το αντιστάθμισμα του ενεργειακού κόστους και του πιθανού κόστους της παραβίασης των συμφωνητικών επιπέδου υπηρεσιών. Αξιολογούμε την προσέγγισή μας και τη συγκρίνουμε με μια υπερσύγχρονη πολιτική δρομολόγησης και διαχείρισης εικονικών μηχανών. Χρησιμοποιούμε χαρακτηριστικά για τους υπολογιστικούς κόμβους που προέρχονται από τη μελέτη εμπορικών διαθέσιμων διακομιστών. Επιπλέον, για τα πειράματά μας χρησιμοποιούμε πραγματικές τιμές για την τιμή της ενέργειας και την τιμή χρέωσης των εικονικών μηχανών. Παραθέτουμε αποτελέσματα με σταθερή και δυναμικά μεταβαλλόμενη τιμή ενέργειας. Τα αποτελέσματά μας δείχνουν κέρδη ως και 17.24% και 17.72% για το συνολικό κόστος και την ενέργεια αντίστοιχα, σε σύγκριση με ανταγωνιστική πολιτική αιχμής που συνδυάζει τη δυναμική κλιμάκωση της τάσης και της συχνότητας του επεξεργαστή με την ανακατανομή των εικονικών μηχανών.

Επιπλέον αξιολογούμε την επιλογή μεταξύ της ενεργειακής αποδοτικότητας και της ενδεχόμενης υπερφόρτωσης των υπολογιστικών κόμβων, στα πλαίσια της διαχείρισης της ανακατανομής εικονικών μηχανών με κοστο-κεντρικό τρόπο. Πιο συγκεκριμένα, εισάγουμε ένα μοντέλο που χρησιμοποιεί πληροφορία που συλλέγεται κατά τη διάρκεια εκτέλεσης για να διαχειριστεί το φόρτο εργασίας των υπολογιστικών κόμβων, αναγνωρίζοντας τη συσχετιζόμενη συμπεριφορά για έργα που εκτελούνται στον ίδιο υπολογιστικό κόμβο. Καθώς οι απαιτήσεις των εικονικών μηχανών δεν είναι εκ των προτέρων γνωστές, είναι σημαντικό να λαμβάνουμε υπόψη την πρόσφατη συμπεριφορά τους ώστε να συγκλίνουμε στον επιθυμητό φόρτο για τον κάθε κόμβο. Συνδυάζουμε την προσέγγιση της μείωσης των περιθωρίων ασφαλείας για την τάση του επεξεργαστή με τη δυναμική κλιμάκωση της τάσης και της συχνότητας. Δείχνουμε πως η πολιτική μας επιτυγχάνει αποτελέσματα ταυτόσημα ή και καλύτερα σε σχέση με 2 πολιτικές αιχμής που συνδυάζουν την ανακατανομή των εικονικών μηχανών με τη δυναμική κλιμάκωση της συχνότητας και της τάσης - με τη δεύτερη ανταγωνιστική πολιτική να αξιοποιεί και αυτή την τεχνική μείωσης των περιθωρίων ασφαλείας για την τάση του επεξεργαστή. Μια σημαντική διαφορά με τις προηγούμενες πολιτικές είναι ότι η μέθοδός μας δεν απαιτεί την εκ των προτέρων – μη ρεαλιστική – ανάλυση των χαρακτηριστικών του φόρτου των εργασιών.

Επίσης μελετούμε την αξιοποίηση των μειωμένων περιθωρίων ασφαλείας, αυτή τη φορά στις παραμέτρους λειτουργίας της μνήμης, και συγκεκριμένα στην τάση και το ρυθμό ανανέωσης. Σε αντίθεση με τους επεξεργαστές, όπου ένα λάθος κυρίως μεταφράζεται σε απότομη διακοπή λειτουργίας του συστήματος (crash), για τις μνήμες μελετούμε το φαινόμενο πιθανών ποινών λόγω των αυτοδιορθούμενων μονού- και ανιχνεύσιμων διπλού-bit σφαλμάτων. Όπως και κατά τη μελέτη για τους επεξεργαστές, εισάγουμε ένα αναλυτικό μοντέλο που εφαρμόζεται ανεξαρτήτως της αρχιτεκτονικής και μια καινοτόμα πολιτική διαχείρισης του φόρτου εργασίας και της διαμόρφωσης των παραμέτρων του συστήματος. Το μοντέλο μας λαμβάνει υπόψιν του τη διαχείριση των εικονικών μηχανών και τη ρύθμιση της μνήμης. Η πολιτική που εισάγουμε, αξιοποιεί το μοντέλο ώστε να εκμεταλλευτεί τις δυνατότητες μείωσης των ορίων ασφαλείας στις παραμέτρους λειτουργίας των μνημών, σε συνδυασμό με την ανακατανομή των εικονικών μηχανών. Δεδομένου ότι τα μελλοντικά υπολογιστικά συστήματα αναμένεται να ενσωματώνουν μεγάλη χωρητικότητα μνήμης DRAM, μελετούμε το πρόβλημα για διάφορα μεγέθη μνήμης. Και σε αυτή την περίπτωση, αξιολογούμε την προσέγγισή μας σε μεγάλη κλίμακα σε περιβάλλον προσομοίωσης, χρησιμοποιώντας πραγματικά χαρακτηριστικά των κόμβων και πραγματικές τιμές για την τιμή της ενέργειας και το κόστος των εικονικών μηχανών. Τα αποτελέσματά μας δείχνουν πως κέρδος της τάξης του 29.53% και 34.84% σε κόστος και ενέργεια αντίστοιχα μπορεί να επιτευχθεί, σε σύγκριση με ανταγωνιστική πολιτική αιχμής που εκμεταλλεύεται την ανακατανομή των εικονικών μηχανών. Τα κέρδη ενισχύονται καθώς κινούμαστε προς κόμβους με μεγαλύτερη ποσότητα μνήμης.

Τέλος, μελετούμε την επιρροή της ετερογένειας και της προσεγγιστικής υπολογιστικής στο ενεργειακό αποτύπωμα εφαρμογών υπολογιστικής υψηλών επιδόσεων. Η ποιότητα του αποτελέσματος δεν επηρεάζεται με τον ίδιο τρόπο από όλους τους υπολογισμούς του προγράμματος, επομένως μπορούμε να εκτελέσουμε επιλεγμένους υπολογισμούς προσεγγιστικά για να βελτιώσουμε την ενεργειακή αποδοτικότητα. χωρίς να πλήττεται σε μεγάλο βαθμό η ποιότητα του αποτελέσματος. Σε αυτή τη μελέτη, χρησιμοποιούμε 4 εφαρμογές από διαφορετικούς τομείς, που κυμαίνονται από μεγάλες εφαρμογές μέχρι υπολογιστικούς πυρήνες. Τροποποιούμε αυτές τις εφαρμογές για να εκμεταλλευτούμε τόσο την ετερογένεια όσο και την προσεγγιστική υπολογιστική. Αξιολογούμε την προσέγγισή μας σε ετερογενή συστήματα (εξοπλισμένα με επεξεργαστές και κάρτες γραφικών) και αποτιμούμε τα οφέλη της ετερογένειας και της προσεγγιστικής υπολογιστικής, λαμβάνοντας υπόψιν την επιλογή με-

ταξύ της κατανάλωσης ενέργειας και της ποιότητας του αποτελέσματος. Τα αποτελέσματά μας δείχνουν πως η ετερογένεια και η προσεγγιστική υπολογιστική μπορούν να προσφέρουν συνδυασμένα ενεργειακά κέρδη.

Λέξεις Κλειδιά

Συστήματα Υπολογιστικής Νέφους, Συστήματα Υπολογιστικής Υψηλών Επιδόσεων, Ενεργειακή Αποδοτικότητα, Ελαχιστοποίηση κόστους, Πολιτικές Υπολογιστικής Νέφους, Δυναμικός όγκος εργασίας, Ετερογένεια, Προσεγγιστική Υπολογιστική

Table of contents

Abstract	xiii
Abstract in Greek	xvii
Table of contents	xxiii
List of figures	xxvii
List of tables	xxix
1 Introduction	1
1.1 Problem	1
1.2 Motivation	3
1.3 Contributions	8
1.3.1 Exploration of CPU and DRAMs operation at reduced margins for cloud infrastructures	8
1.3.2 Enabling adaptive policies	10
1.3.3 Exploiting the combination of heterogeneous and approximate com- puting	12
1.3.4 Outline	13
2 Background	15
2.1 CloudSim	15
2.1.1 Nodes and VM applications	15
2.1.2 VM scheduling	16
2.2 Centaurus programming model	17

3	Modelling the benefits and cost of Voltage and Frequency Scaling	19
3.1	Models	20
3.1.1	Assumptions	20
3.1.2	SLA violation penalty	21
3.1.3	Error Modeling	22
3.1.4	Energy Consumption Model	22
3.1.5	Energy Reduction Methodologies	23
3.1.6	Total cost	25
3.2	Evaluation	25
3.3	Related Work	27
4	Exploitation of CPU Voltage Margins for Cost Reduction at Cloud Infrastructures	29
4.1	Model	30
4.1.1	Nodes and CPU operating points	31
4.1.2	Application workload – VMs	32
4.1.3	VM scheduling	32
4.1.4	CPU load and CPU capacity allocation to VMs	33
4.1.5	Failures due to operation at reduced margins	33
4.1.6	SLA violation cost	34
4.1.7	Energy cost	35
4.1.8	Overall operating cost estimation	35
4.2	VM Scheduling and System Configuration Policy	36
4.3	Hardware Characterization	39
4.3.1	Voltage margin characterization	39
4.3.2	Failure probability	41
4.3.3	Power consumption	41
4.3.4	Performance degradation due to frequency scaling	42
4.4	Experimental Evaluation	43
4.4.1	Target node utilization sensitivity analysis	44
4.4.2	XM-VFS behavior at the baseline failure probability	46
4.4.3	XM-VFS behavior at different failure probabilities	48
4.4.4	XM-VFS behavior with varying energy pricing	49

4.5	Related Work	50
5	Adaptive utilization threshold for Cloud Computing	55
5.1	Background	56
5.1.1	System model	58
5.1.2	VM allocation approach	59
5.2	Problem Modeling	60
5.2.1	Target CPU utilization and VM evictions	60
5.2.2	Load estimation	61
5.2.3	Energy cost estimation	62
5.2.4	SLA violation cost estimation	63
5.2.5	Total estimated node cost	64
5.3	Adaptive target CPU utilization	64
5.3.1	System model	64
5.4	VM Allocation and Node Configuration Policy	67
5.5	Experimental Evaluation	69
5.5.1	Experimental Setup	70
5.5.2	Hardware parameters	71
5.5.3	Threshold and cost convergence	72
5.5.4	<i>ATUC</i> total cost of operation	74
5.5.5	Comparison of <i>ATUC</i> with state-of-the-art policies	75
5.6	Related Work	77
6	DRAM operation at reduced margins for profit increase of cloud providers.	79
6.1	Problem Modeling	80
6.1.1	Nodes and DRAM operating points	82
6.1.2	Workload – VMs	82
6.1.3	VM scheduling	83
6.1.4	Errors due to relaxed refresh rate of DRAMs	84
6.1.5	CPU load and CPU capacity allocation to VMs	85
6.1.6	SLA violation cost	86
6.1.7	Energy costs	86
6.1.8	Operating cost	87

6.2	Scheduling and System Configuration Policy	87
6.2.1	Approximations	88
6.2.2	VM Scheduling and Node Configuration – RM-DRAM	88
6.3	Characterization and Modeling of DRAM Power and Error Behavior	91
6.3.1	DRAM error modeling	91
6.3.2	Characterization Infrastructure	92
6.3.3	Modeling DRAM Power and Power at the Plug	94
6.4	Experimental Evaluation	95
6.4.1	Simulation Setup	95
6.4.2	Results and Discussion	97
6.5	Related Work	99
7	Heterogeneity and approximate computing for energy efficiency	101
7.1	Applications	102
7.1.1	Smallpt	102
7.1.2	LULESH	103
7.1.3	PBPI	104
7.1.4	Conjugate Gradient	105
7.2	Evaluation	105
7.2.1	Smallpt	106
7.2.2	LULESH	107
7.2.3	PBPI	108
7.2.4	Conjugate Gradient	108
7.3	Related Work	109
8	Conclusions	111
	References	115
	APPENDICES	127
A	Related Publications	129
B	Contribution to Joint Publications	131

List of figures

1.1	Available margins for: (α') Intel Xeon E3-1220 v5 voltage, (β') DDR3 DRAM voltage and (γ') DDR3 DRAM RP.	4
1.2	Cost gains due to the exploitation of reduced CPU margins.	5
1.3	CPU pressure of different VMs w.r.t their contractual (SLA) maximum CPU requirements, over consecutive periods.	6
1.4	Output of a global illumination renderer that produces high quality scenes .	7
3.1	Operator cost reduction using Voltage Scaling.	28
3.2	Operator cost reduction using Frequency Scaling.	28
3.3	Operator cost reduction using Voltage and Frequency Scaling.	28
4.1	Voltage margins characterization for the Skylake (<i>left</i>) and the X-Gene 3 CPUs (<i>right</i>).	40
4.2	Classes of applications, in terms of performance sensitivity to frequency reduction for Skylake (<i>left</i>) and X-Gene 3 (<i>right</i>).	42
4.3	Distribution of energy prices during a single day.	44
4.4	Target utilization threshold sensitivity analysis for the Intel (<i>left</i>) and ARM (<i>right</i>) -based nodes.	45
4.5	Parametric analysis of cost w.r.t. probability of failure with low, average and high energy prices for PPR and FPR SLAs on both platforms.	46
4.6	Percentage of nodes at reduced margins for both platforms.	48
4.7	Percentage of the nodes at reduced margins with variant energy prices for different failure probabilities for the Intel and ARM platforms.	51
4.8	Normalized cumulative workload of the VMs and energy prices per period.	51
5.1	Overview of the proposed approach.	59

5.2	CPU pressure of different VMs w.r.t their contractual (SLA) maximum CPU requirements, over consecutive periods.	65
5.3	Cumulative CPU load (normalized w.r.t the maximum cumulative CPU load) variation in time (across different periods).	70
5.4	$\overline{u^{target}}$ per period for different initial values of u^{target} (0.5 – 1.0).	72
5.5	Normalized cost per period for initial values of target utilization of 0.5, 0.7 and 1.0.	73
5.6	Normalized (total) cost for different initial values of u^{target}	74
5.7	Cost comparison of <i>ATUC</i> with <i>LrMmt</i> and <i>XM-VFS</i>	76
6.1	Overview of the proposed approach.	80
6.2	Modeling $CE_{i,m,p}$ variation for different DRAM chips.	93
6.3	a) The maximum and average DRAM power; b) Scaling of DRAM power savings with the refresh period; c) Scaling of DRAM power savings (at the plug) with memory size.	94
6.4	Relative cost (left y-axis, bars, normalized w.r.t. <i>LrMmt</i> at the same memory capacity) and relative energy consumption (right y-axis, lines, normalized w.r.t. <i>LrMmt</i> at 512GB), for effectively equal node utilization. Left y-axis starts at 0.4.	96
6.5	Relative cost (left y-axis, bars, normalized w.r.t. <i>LrMmt</i> at the same memory capacity) and relative energy consumption (right y-axis, lines, normalized w.r.t. <i>LrMmt</i> at 512GB), for the same target node utilization. Left y-axis starts at 0.4.	96
6.6	Percentage of nodes configured to each refresh period setting (<i>RP</i>) for different DRAM capacities and temperatures.	99
7.1	(α') Smallpt evaluation, (β') LULESH evaluation	106
7.2	(α') Smallpt fully accurate output, (β') Smallpt fully approximate output	107
7.3	(α') PBPI evaluation, (β') CG evaluation	108

List of tables

3.1	Summary of variables.	20
4.1	Summary of variables.	30
4.2	Target hardware platforms.	39
5.1	Summary of variables.	56
5.2	Intel Xeon E3-1220 v5 and ARM-based Ampere Computing X-Gene 3 op- erating points.	71
6.1	Summary of variables.	81
7.1	List of benchmark applications and their characteristics.	102

Chapter 1

Introduction

Modern datacentres process and exchange a huge amount of data and requests, and the demand for resources increases significantly. This demand may also increase the energy consumption of the nodes in order to fulfill the requirements of the customers, affecting in turn the operational cost. Providers are also bound to Service Level Agreements (SLAs), and the breach of such agreements leads to penalties which also increase the cost. Improving the energy efficiency of the datacentres is far from trivial, as there is a trade-off between the energy cost and the Quality of Service (QoS).

As a consequence, current and next-generation systems need to operate under strict energy constraints offering at the same time the required performance to the customers. Therefore, new solutions that promise energy gains considering the trade-off between energy consumption and QoS need to be investigated.

1.1 Problem

It is estimated that the energy consumption of datacentres will increase from 200 TWh in 2016 to 2967 TWh in 2030 [97], and minimizing energy consumption has become a critical concern in computing, due to economic and environmental reasons. The energy cost of running a datacentre is very high, as the energy consumption of a representative datacentre is equal to 25,000 households [37], and the respective costs are not expected to improve.

Nodes in modern datacentres are equipped with multi-core processors and high amounts of Dynamic Random Access Memory (DRAM) to meet the requirements of the services and execute computing applications efficiently. These two components contribute the most

significant portion to the total power consumption of the nodes. Namely, CPUs account for up to 60% of the total power consumption of the compute nodes [33], while DRAMs are rapidly becoming a major power consumer for servers, already contributing a significant fraction (around 30%) of the total power consumption of the node [30,31].

Voltage and Frequency Scaling (VFS) [8] tries to address the problem of excessive CPU power consumption, enabling the joint manipulation of the CPU supply voltage (V_{dd}) and frequency (f) among different (V_{dd}, f) points. This way, a node can be configured to a more power-efficient operating point, reducing both f and the respective V_{dd} however, this is possible only when the system identifies opportunities (slack) to do so without penalizing the performance of the workload to an unacceptable extent.

On the software side, and particularly in the context of Cloud Computing, virtual machine (VM) consolidation [19,28,29] is a common technique for reducing the power consumption of a datacentre. VM consolidation triggers VM relocation to nodes when necessary, aiming to reduce the number of active nodes avoiding any performance loss, thus minimizing energy consumption. However, this technique requires the prediction of the VM resource requirements, otherwise the nodes may suffer from underloading or overloading. Therefore, VM consolidation should be applied in a sophisticated manner in order to avoid any increased energy cost or potential SLA violation penalties.

A typical hardware/software approach to improve power efficiency is heterogeneous computing, often exploiting Graphics Processing Units (GPUs) as accelerators. In contrast to conventional CPU-like architectures, GPUs offer massive, partially asynchronous parallel execution through many computational cores. Although GPUs consume more power than conventional processing units, they are superior in terms of performance per Watt, and they can significantly reduce the execution time and the energy consumption of applications, especially with the existence of sufficient parallelism and GPU-friendly computational and memory access patterns. However, heterogeneous computing requires extra programming effort.

Another, more aggressive, software approach towards improving the energy efficiency of applications is approximate computing. All parts of an application are often assumed to have the same significance (importance) for the quality of the output however, approximate computing disrupts this conservative approach. Approximate computing is based on the observations that: a) there are applications, such as computer vision, simulations, etc, that can

tolerate some loss of precision of their calculations, and b) not all the parts of an application contribute the same towards the quality of the output results. Approximate computing relaxes or even drops some calculations of the applications, introducing a trade-off between energy consumption, performance and quality of output. As in the case for heterogeneity, approximate computing also requires extra programming effort.

1.2 Motivation

CPU manufacturers introduce voltage guardbands on top of the required voltage for each frequency step, in order to guarantee stable operation under extreme combinations of manufacturing variability, aging effects, workload stress, environmental conditions, etc. These guardbands are conservative, as the worst-case combination of those adverse conditions will occur very rarely, if at all, during the life of each chip, and they increase the power consumption of the CPU by about 35% [6]. Hence there are significant opportunities for energy savings by reducing these guardbands in an educated manner [16,21] and supplying the CPU with less voltage for the respective operating frequency (undervolting).

Voltage guardbands exist also on DRAMs, and they are also chosen conservatively, based on pessimistic assumptions. Also, to address the inherently limited cell retention time, modern DRAMs employ an *Auto-Refresh* mechanism that periodically recharges all cells at a specific *refresh period* (RP). Therefore, DRAMs adopt frequent refresh cycles to address the increased variability of cell retention time within, and across DRAM chips [34]. Conventionally, the refresh period is determined based on the worst-case retention time across all manufactured cells, which is estimated assuming extremely rare, worst-case combinations of operating conditions (e.g. extremely high temperature and worst-case access and stored data patterns), also including an additional non-negligible safety guardband [35]. Based on that philosophy, DDR technologies adopt the same refresh period (i.e. 64 ms , or 32 ms in case of high temperature) even if, in reality, most cells may have much higher retention time. Such a pessimistic refresh period leads to considerable power and performance overheads, which are expected to worsen as DRAM densities increase. It has also been projected that future 64 Gb DRAM chips will spend 35% of the memory power and 25% of the available DRAM chip bandwidth on refreshes only [36].

Figure 1.1 presents the extent of these guardbands derived from real systems. More specif-

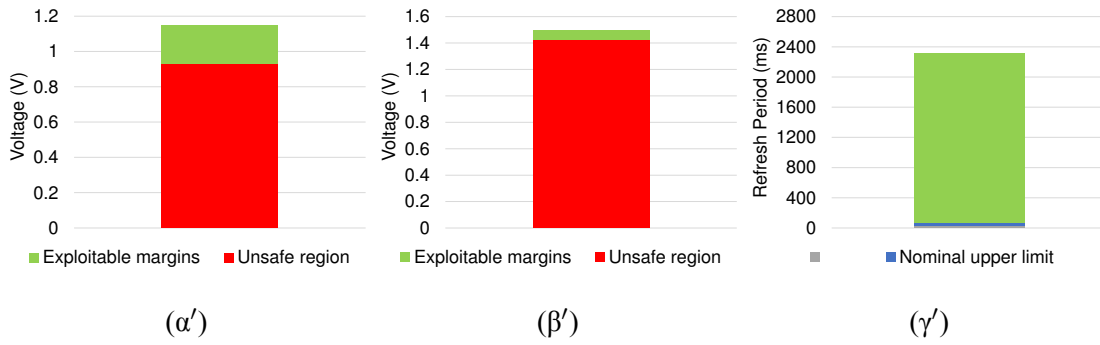


Figure 1.1: Available margins for: (α') Intel Xeon E3-1220 v5 voltage, (β') DDR3 DRAM voltage and (γ') DDR3 DRAM RP.

ically, the left figure illustrates the degree of undervolting for the Intel Xeon E3-1220 v5 processor, when this processor operates at the maximum frequency (3.3 GHz). The nominal voltage for the maximum frequency is 1.146 V. However, this processor has a wide voltage margin (green area) and the reduction of the voltage from 1.146 V to 0.929 V offers important energy gains. The middle and right figures represent the voltage and RP margins of the DRAMs for the DDR3 technology, respectively. DRAMs are supplied with 1.5 V and have a RP of 64 ms nominally. We observe that these values are also pessimistic offering opportunities for more energy gains exploiting the voltage (1.5 V to 1.428 V) and the RP (64 ms to 2283 ms) margins of the DRAMs.

However, exploiting these margins may make the system more vulnerable to errors, affecting – even if slightly – system stability and increasing the probability of failures, especially for scale-out deployments. Node instability may lead to SLA violations, which, in turn, lead to penalties and thus to reduced profits for the provider of the computing infrastructure. Therefore, dealing with the trade-off between energy cost and SLA violation penalties is very important when exploiting the configuration at reduced margins, in order to avoid SLA violation penalties due to failures and increase the profit of cloud infrastructure providers.

Figure 1.2 depicts the effect on the total cost for lower than nominal voltage values when the processor operates at the maximum frequency. In order to model this cost we examine the trade-off between energy gains and potential SLA violation penalties using real parameters from the Intel Xeon E3-1220 v5 processor for the estimation of the power consumption. The green line represents the energy cost (power consumption is measured at the plug), the blue line is the SLA violation penalties due to failures, and the black line illustrates the total cost which consists of the sum of the energy cost and the SLA violation penalties. Values are

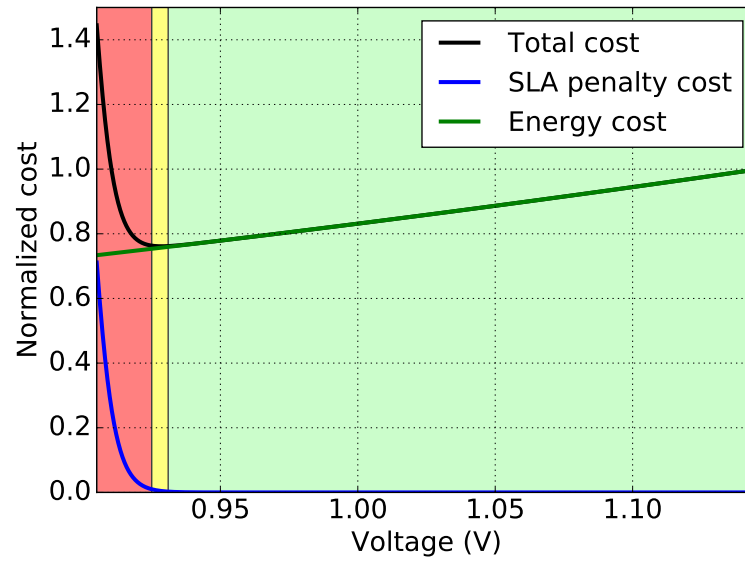


Figure 1.2: Cost gains due to the exploitation of reduced CPU margins.

normalized w.r.t. the total cost at the nominal operating point. We observe, that by lowering the voltage of the CPU (moving right to left along the horizontal axis in the figure), there is a large area (green area) that can offer energy and cost gains, where the failure rate increase is negligible, and the occurrence of errors is, consequently, very rare. Moving to lower voltage values, we may enter the yellow area, where the failure rate increases slightly, and the cost increases too due to SLA violations. Finally, there is the red area, where the failure rate increases significantly, and it should be avoided due to SLA violation penalties.

We observe that the cost gains before entering the yellow area w.r.t. the nominal configuration can be up to 24.26%. Our target is to explore the opportunities for reducing the energy and the total cost, exploiting the benefits of the green area which offers significant energy and cost gains with a very rare occurrence of errors. In any case, the operator should avoid moving into the red area, where the SLA violation penalties due to errors outweigh the benefits of the reduced margins configuration. Hence, new VM scheduling and node configuration policies are necessary in order to exploit these margins in an educated manner and avoid or at least control potential SLA violation penalties.

Moreover, scheduling, configuration and resource management policies for the cloud often use static thresholds to drive their decisions [39], with node target utilization thresholds being a typical example. This popular approach has low design complexity, however (a) selection of the optimal static threshold requires *a priori* knowledge of the workload, which may not always be realistic, (b) it fails to capture inter-node variations of workload charac-

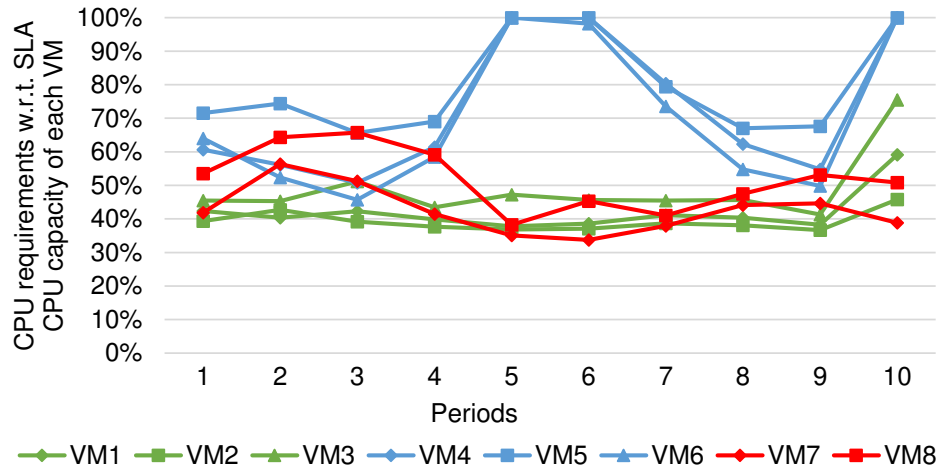


Figure 1.3: CPU pressure of different VMs w.r.t their contractual (SLA) maximum CPU requirements, over consecutive periods.

teristics at any given time, (c) it fails to capture system-wide workload variations in time, and (d) the value of the optimal threshold is also sensitive to hardware characteristics and configuration.

These shortcomings can be addressed by designing policies that dynamically adjust the target load threshold of the nodes. These policies decide the target load threshold of the node, with the goal of increasing energy efficiency, by analyzing the workload characteristics observed in the past. However, the selection of such a threshold is far from trivial, as an SLA violation may occur even when operating at a very pessimistic yet realistic threshold. Figure 1.3 illustrates such an example presenting the percentage of the CPU requirements with respect to the contractually promised CPU capacity of eight VMs from Google traces [26] through time (time is divided into *periods*, and a typical duration of the period is 300 seconds [19]). Each line corresponds to a different VM. One can identify clusters of VMs with similar, positively correlated variations (according to their percentage of the CPU requirements) (lines of the same color). Moreover, VMs in the blue and green cluster are characterized by a rapid increase of their requested CPU capacity at period 10, while the red cluster appears to have the opposite trend compared with the blue one (anti-correlated).

Hence, the identification of positively correlated VMs should encourage more conservative values of the target CPU utilization, while the co-execution of anti-correlated VM clusters on the same node can protect against overloads due to dynamic workload changes: when one cluster increases its requests, its counterpart tends to decrease them, thus balancing the total load on the host node. This allows for more aggressive (higher) CPU utilization targets, and a

more energy-efficient usage of the available resources. According to our experiments, if the correlation alone is overlooked, the cost achieved by a scheduling policy that determines the utilization target at runtime is 24.52% higher.

For HPC systems, an approach that can significantly reduce the energy consumption is the combination of heterogeneity with approximate computing. Approximate computing techniques can benefit applications that operate under tight energy budgets, yet they can tolerate / recover from inaccurate computations. Hardware accelerators, such as GPUs, can in turn be designed to natively support different approximations. Therefore, this combination opens up new possibilities for power/energy management. As a result, the energy footprint of the applications can be reduced, and large-scale and HPC centers could significantly cut down their power costs.

Figure 1.4 illustrates the output of a global illumination renderer that produces high-quality scenes. The approximate version ignores the diffuse reflections, thus it is less computationally intensive compared with the accurate execution which takes into account all types of the reflection destination point. Our experiments quantify the energy gains from the approximations to up to 71.4%, while the differences of the output between the accurate and approximate versions are minor.

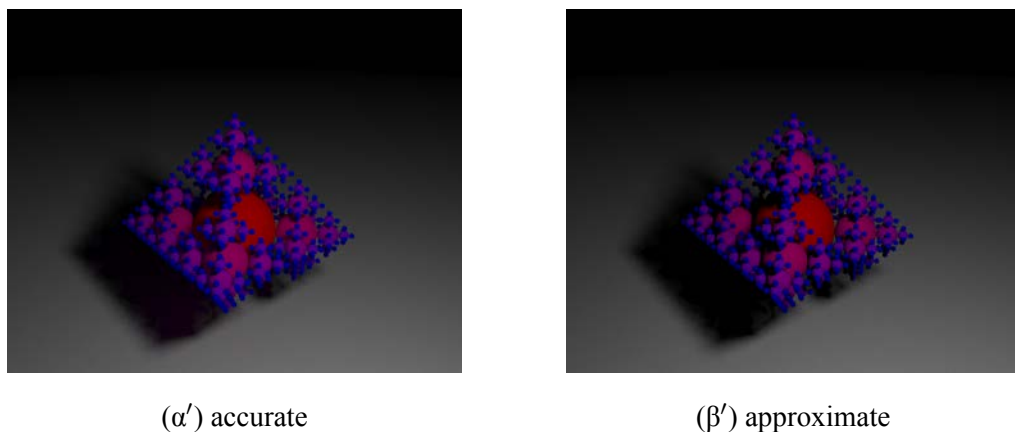


Figure 1.4: Output of a global illumination renderer that produces high quality scenes

1.3 Contributions

1.3.1 Exploration of CPU and DRAMs operation at reduced margins for cloud infrastructures

We study the reduction of the margins of operating parameters (voltage for CPU, voltage and RP for DRAMs), based on the observation that there are opportunities for reducing the energy cost significantly. Operating parameters' margins used to guarantee correct operation are unnecessarily wide for real-world execution scenarios. However, the excessive reduction of those guardbands may lead to node failures which will trigger SLA violation penalties and will increase the cost. Therefore, we provide an extensive theoretical and experimental study to address the following questions:

- "Is it possible to identify energy-efficient configurations for the CPU in the multi-parametric configuration space through modeling?"
- "Can operation at reduced CPU voltage margins increase the profit of Cloud infrastructure providers?"
- "Similarly to CPUs, can operation at reduced margins of the DRAMs operating (voltage and RP) parameters increase the profit of Cloud infrastructures providers?"

Is it possible to identify energy-efficient configurations for the CPU in the multi-parametric space through modeling?

One of the most important steps to increase the profit for Cloud and Edge operators is to identify the proper configuration of the CPU. For our study, we examine different configurations of the CPU using the three following techniques: a) VFS, b) voltage overscaling (undervolting), and c) frequency overscaling (overclocking). These techniques may, however, lead to node failures, whereas two of them (VFS and overclocking) also affect the performance. Voltage and frequency overscaling make the processor more vulnerable to timing errors. VFS can also affect system reliability, although in a less profound way; operating to lower voltage/frequency steps makes CPU cells more vulnerable to lower energy transient upsets, like the effects of cosmic particles. As a result, the quest to reduce energy consumption may affect the quality of service (QoS) provided by the system, triggering SLA violation penalties and reducing the profit margins.

In this work, for all three approaches, we estimate the cost for the operator due to energy consumption and potential SLA violation penalties and study its sensitivity to different parameters. For our evaluation, we use tasks with various deadlines. To the best of our knowledge, this is the first effort to study the operation of edge/cloud nodes outside nominal configuration from a cost minimization perspective.

Can operation at reduced CPU voltage margins increase the profit of Cloud infrastructure providers?

The exploitation of the reduced voltage margins of the CPU may reduce the energy consumption of cloud infrastructures however, this should be done in an educated manner in order to avoid any potential SLA violation penalties. Therefore, we investigate how such margins can be exploited by infrastructure operators, by selectively undervolting nodes, at the controlled risk of inducing failures and activating SLA violation penalties. Similarly to the previous section, we identify energy-efficient configurations for the CPU. However, in this work, we introduce a complete model that captures the most important aspects that drive VM management and system configuration decisions. More specifically, apart from node failures due to configuration to non-nominal settings, this model considers re-scheduling of the VMs to nodes and potential node overloads for the SLA penalties. We also introduce a novel scheduling and system configuration policy based on the model. The policy exploits the reduced voltage margins of the CPU on top of both VFS and VM consolidation. Finally, in this work, we evaluate our approach through simulations to quantify the cost reduction, considering the energy consumption and potential SLA violations, and we compare our approach with a state-of-the-art policy that combines VFS with VM consolidation.

For our evaluation, we characterize two commercially available server-class systems. These systems have different processor architectures, and we quantify their voltage margins, power consumption and computational capacity under different configurations. We take into account that a datacentre typically runs a variety of jobs with different performance characteristics (e.g., I/O-bound or CPU-bound applications) which do not behave similarly when applying VFS. We also use realistic values for the energy price (we use both a constant value and variant values of the energy price throughout the day) and the VM pricing, and we provide results for the energy gains at the plug rather than the processor itself. To the best of our knowledge, this is the first work to study the operation of a cloud infrastructure beyond the

nominal configuration of server machines, considering the VM scheduling and node configuration problem using real hardware attributes.

Can operation at reduced margins of the DRAMs operating parameters (voltage and RP) increase the profit of Cloud infrastructures providers?

We examine the trade-off between energy cost and potential SLA violation penalties, however from the perspective of reducing the margins of operating parameters of the DRAMs. As we discussed earlier, the voltage supply margins for DRAMs are also unnecessarily wide. These margins exist for the *RP* as well. As in the case for the CPU, the excessive reduction of these margins may lead to node crashes. Apart from node failures, this reduction may also increase the load of the node, as DRAMs have mechanisms for handling errors, which will require a portion of the capacity of the processor. Hence, any non-judicious relaxation of these parameters may again result in DRAM errors which may, in turn, violate the SLA between the provider and the customer, thus triggering penalties.

For this purpose, we reformulate our previous models to capture also the additional CPU load and the failure probability of the nodes due to DRAM errors. We also introduce a scheduling and node configuration policy, based on our model, that employs cost-aware (rather than threshold-based) VM consolidation and exploits the reduced margins of DRAMs to increase the operator profit by considering the trade-off between the cost of energy consumption and potential SLA violations. We extract the parameters used in the simulation (particularly power consumption and error rates) through the characterization of a commercially available server. We perform simulations to evaluate the effectiveness of our approach against a state-of-the-art policy that exploits VM consolidation. We also experiment with different values for the temperature and the memory capacity of the nodes. To the best of our knowledge, this is the first work to propose a policy that considers DRAM operation at reduced margins on top of VM consolidation to reduce the cost of cloud infrastructure operators.

1.3.2 Enabling adaptive policies

We study the calculation of adaptive target utilization thresholds for the nodes of cloud infrastructures, based on the fact that static target utilization thresholds fail to capture the workload variations and selecting an optimal static target utilization is unrealistic for real-

world workloads. We focus on answering the following question: "Can a policy that calculates the target utilization of the nodes at runtime outperform policies that assume *a priori* knowledge of the workload using offline characterization to derive an optimal static target utilization?".

Can a policy that calculates the target utilization of the nodes at runtime outperform policies that assume *a priori* knowledge of the workload using offline characterization to derive an optimal static target utilization?

As CPU is a key contributor to the energy consumption of the nodes, increasing the utilization of CPU resources on active nodes is a key step towards energy efficiency. However, this is a challenging undertaking, as the workload can vary significantly among the nodes and over time, exposing operators to the risk of overcommitting the CPU, thus affecting the QoS of workloads.

In our approach, we extend our previous model (CPU reduced voltage margins model) to apply cost-driven VM eviction from the nodes and calculate the target utilization. More specifically, the extended model predicts the load of the node in the following period, as well as the associated cost, in order to take the most cost-efficient decision in terms of VM scheduling and node configuration. Although we consider each node separately, we estimate the cost for the whole datacentre, thus our approach is both effective at the datacentre granularity and scalable. We also introduce a model that uses runtime information to adaptively decide the target utilization of the nodes to control their load. The model identifies correlated behavior among colocated workloads and considers such patterns in its estimations. Our policy is based on these models to drive VM management and node configuration decisions to increase the profit of the datacentre operator by carefully considering the trade-off between the reduction in energy consumption and potential costs due to SLA violation penalties. In order to further reduce the operating cost we also consider the reduced voltage margins of the CPU on top of VFS and VM consolidation. We improve our previous approach (exploitation of CPU voltage margins) by employing these adaptive utilization targets rather than static thresholds, and we present an extensive evaluation based on simulations comparing our policy with our previous and a state-of-the-art policies that use an optimal static target utilization derived through offline characterization of the workload. To the best of our knowledge, this is the first work that combines per node target CPU utilization adaptivity with VFS and relaxed

CPU voltage guardbands in a cost-centric approach, to increase the profit of cloud infrastructure providers exploring the trade-off between energy cost and potential SLA violation penalties.

1.3.3 Exploiting the combination of heterogeneous and approximate computing

We study the combined effects of heterogeneous and approximate computing, based on the facts that: a) accelerators, such as GPUs, are superior in terms of performance per Watt, and b) result quality is not equally affected by all computations of an application. Our study comes to answer the following questions:

- "Can we apply approximate computing on HPC applications and provide a trustful output?"
- "What are the gains of the combination of heterogeneous and approximate computing for HPC applications?"

Can we apply approximate computing on HPC applications and provide a trustful output?

Although approximate computing may improve the energy efficiency of applications, it can also affect the quality of output. To address this issue, we introduce 4 applications, which are part of the AcHEe benchmark suite (Approximate Computing and Heterogeneity for Energy efficiency). AcHEe is developed using mainly OpenCL nomenclature, and it can be executed in any architecture that supports OpenCL. It comprises a mix of both kernels and larger-scale applications from diverse domains, and the applications are modified to support both heterogeneity and approximate computing. Moreover, we provide an accurate and approximate version of the computationally intensive parts, carefully chosen to improve energy efficiency, but yet avoiding any significant loss of quality of the output.

What are the gains of the combination of heterogeneous and approximate computing for HPC applications?

Heterogeneity may improve the energy efficiency of an application, as GPUs are equipped with many computational cores and they offer a greater ratio of performance per Watt than

conventional CPUs. These accelerators may have a slightly higher power consumption, but their massive computation capacity may decrease the execution time for the application, reducing the energy footprint significantly as well. Moreover, the combination of heterogeneity with approximate computing can pave the way for instructing the approximations to the device that it is most favored by it, in order to improve the energy efficiency of HPC applications. For this study, we use again 4 application from the AcHEe benchmark suite, and we illustrate the energy gains of each application for different architectures and different levels of approximations.

1.3.4 Outline

Chapter 2 summarizes the technical background assumed by the following Chapters.

Chapter 3 provides a study to identify configurations for operating the CPU at non-nominal configurations to improve the energy efficiency. We present our approach using VFS, undervolting and overclocking, and for each one of these techniques, we provide a model that estimates the energy consumption and potential SLA violation penalties due to node failures.

In Chapter 4, we introduce a novel scheduling and system configuration policy (*XM-VFS*) that exploits the reduced voltage margins of the CPU. *XM-VFS* deals with the trade-off between energy consumption and potential SLA violation penalties, and finally, we evaluate our policy through simulations compared with a state-of-the-art policy. In our work, we use both static and variant values for the energy price.

Chapter 5 deals with the problem of dynamic workloads in Cloud Computing. In this work, we consider the correlations between the workload of the VMs of the nodes, and we introduce the *ATUC* policy that takes into account these correlations to allocate the VMs on the nodes. We evaluate our policy through simulations compared with two state-of-the-art policies that utilize a static optimal utilization threshold defined through offline characterization of the workload.

Later, in Chapter 6, we focus on exploiting the reduced operating parameters of the DRAMs to improve energy efficiency, introducing another novel scheduling and system configuration policy (*RM-DRAM*). We evaluate *RM-DRAM* through simulations comparing it against a state-of-the-art policy. We investigate the efficiency of *RM-DRAM* for different values of temperature, and memory capacity of the nodes.

Chapter 7 presents the benefits of combining heterogeneity with approximate computing. We apply this combination in various applications, and we use different approximation techniques. Our results indicate that heterogeneity combined with approximate computing can improve the energy efficiency significantly.

Finally, Chapter 8 concludes the dissertation.

Chapter 2

Background

2.1 CloudSim

The use of real testbeds for the evaluation of system-level Cloud policies can offer realistic results at small scales. However, such testbeds are typically deployed at limited scale, therefore they cannot be used to assess the efficiency and performance of policies at the scale expected in current and emerging cloud infrastructures. Therefore, in our research, we use CloudSim [25] which is an open-source simulator for evaluating Cloud policies. More specifically, CloudSim is a framework that enables modeling, simulation and evaluation of Cloud policies, enabling extensive testing and evaluation of them. The most important aspects that make CloudSim suitable for our research are:

- It provides fully parameterized options for the hardware of the nodes and the VM characteristics.
- It provides management for the full life-cycle of VMs with a VM scheduling mechanism.
- It provides abstract classes for the mechanisms of monitoring VM execution and of VM-to-node mapping, which are easily extensible to enable the experimentation with different policies.

2.1.1 Nodes and VM applications

CloudSim users can create a virtual datacentre that manages a number of nodes where the VMs will be accommodated. The advantage of CloudSim is that users can define several

hardware parameters for the nodes using characteristics from real systems. Such parameters are the processing power of the CPU (CPU capacity), the amount of the memory and the storage of the node.

CloudSim users can also manage the entire life-cycle of a VM: a) provisioning of a node for hosting the VM, b) creation of the VM, c) migration of the VM, and d) destruction of the VM. When a VM is created, it requests a specific amount of the CPU and memory capacity of the node as defined in the SLA. CloudSim will create and host these VMs on hosts that fulfill those requirements. More specifically, VMs are mapped to nodes using a VM allocation policy component, which considers the CPU and the memory capacity of candidate nodes.

2.1.2 VM scheduling

CloudSim divides the total simulation time into smaller periods where scheduling decisions are taken. The user can define both the total simulation time and the duration of the scheduling periods. Real Cloud workloads are dynamic in terms of CPU requirements, and therefore these requirements change over time. CloudSim incorporates mechanisms to estimate the requirements of the VMs at the beginning of each scheduling period and use different policies to re-allocate the VMs – if needed – on the nodes more efficiently. The following steps (also described in [19]) form the basis for the (re-)scheduling of VMs on the available nodes.

1. **Identification of node overloads:** The first step performs re-estimation of the VM requirements to identify potential node overloads for the next scheduling period, which would increase the cost of the datacentre (due to potential SLA violation penalties). These nodes are marked and some of their VMs are selected for migration to reduce the load. CloudSim offers an abstract class (`PowerVmAllocationPolicyAbstract`) that can be easily extended to add more policies for identifying potential node overloads.
2. **VM selection for migration:** At this step, CloudSim selects the VMs that should be migrated from the nodes marked as overloaded to decrease their load. Again, CloudSim implements an abstract class (`PowerVmSelectionPolicy`) that can be extended in a straightforward way to experiment with different policies for selecting VMs for migration.

3. **Node selection for hosting VMs:** After the selection of target VMs for migration, CloudSim provides a function for allocating the migrating VMs to the nodes. Each migrating VM is assigned to the node that will suffer the lowest increase in power consumption due to the migrated VM. Modifying this function, Cloudsim users can experiment different optimization goals when allocating VMs on the nodes.
4. **Identification of underutilized nodes:** As a final step, the identification of underutilized nodes is also critical, as migrating VMs away from these nodes may provide opportunities to power the nodes off.

Our methodology in this dissertation also follows this steps. We introduce extensions and new policies, which we discuss in the next chapters.

2.2 Centaurus programming model

This section briefly describes the Centaurus Programming Model [82] which enables the execution of applications on heterogeneous architectures while at the same time exploiting approximate computing. The programmer that uses the programming model specifies high-level information for the execution of the application. Then, the Centaurus compiler and runtime environment exploit this information to automate the execution of the application on different architectures with different (potentially varying at execution-time) levels of approximation to trade-off among execution time, quality of results and energy efficiency according to the constraints set by the programmer. We use this model in our study in Chapter 7.

The Centaurus programming model adopts a task-based paradigm, using *#pragma* directives to annotate parallelism and approximations. Tasks are implemented as OpenCL kernels [62]. Tasks are organized in task groups, and it is possible to control the level of approximations at the granularity of task groups. Listing 2.1 summarizes the *#pragma* task and *taskwait* directives used for task management.

The *task* directive defines a new task. It specifies a function call which corresponds to an OpenCL kernel (*accurate_task_impl(...)*) which specifies the accurate implementation of the task. The programmer can also define an approximate implementation of the task, with a lower energy footprint, using the clause *approxfun()*.

Listing 2.1: Pragma directives for a task for the Centaurus model.

```
#pragma acl task [approxfun( function )] [significant( expr )] [label("name")] \
    [in( varlist )] [out( varlist )] [inout( varlist )] \
    [device_in( varlist )] [device_out( varlist )] \
    [device_inout( varlist )] \
    [workers( int_expr_list )] [groups( int_expr_list)] \
accurate_task_impl( ... );
#pragma acl taskwait [label("name")] [ratio( double )]
```

The *significant()* clause expresses the significance of the task for the quality of the output. The value ranges from 0.0 to 1.0. If 0.0 or 1.0 are selected, the runtime will always execute the task approximately or accurately, respectively.

The *in()*, *out()* and *inout()* data clauses are used to define the input and output arguments of the task. The runtime uses this information to resolve data dependencies between memory objects and to automate data transfers between different devices in the case of heterogeneous systems. The *device_in()*, *device_out()* and *device_inout()* data clauses are an extension of the previous clauses, and they always force data transfers from / to device.

The programmer uses the *workers()* and *groups()* clauses to define the geometry of the kernel, which follow the semantics of local- and global workgroup size of OpenCL, respectively.

The *label()* clause associates tasks with named task groups. Each group is characterized by a unique string identifier.

The *taskwait* is a synchronization barrier for the execution of the tasks, and by default, it forces all the issued tasks to wait, unless *label()* is defined, where it forces only the task of the specific group to wait.

Finally, the *ratio()* clause defines the minimum percentage of the tasks that will be executed accurately in a taskgroup. It accepts values from 0.0 to 1.0, and 0.0 means that the runtime does not need to execute any tasks accurately, while 1.0 forces the runtime to execute the accurate function of the tasks regardless of their significance.

Chapter 3

Modelling the benefits and cost of Voltage and Frequency Scaling

In this chapter we focus on the question "Is it possible to identify energy-efficient configurations for the CPU in the multi-parametric space through modeling?". More specifically, we study the energy consumption for edge and cloud providers exploiting Voltage and Frequency Scaling (VFS) and the more aggressive techniques of voltage overscaling (undervolting) and frequency overscaling (overclocking) considering potential SLA violation penalties. We compare our approaches against the operation of the processor at the highest nominal operating frequency (f_{max}) and voltage (V_{max}).

For our evaluation, we use the characteristics of a commercial available processor, the Intel Core i7 4820K. Moreover, we use realistic values for the energy price and the prices of the VMs, and finally, we present results with different duration of the tasks. Our results show that we can achieve cost gains up to 13.31% (undervolting approach) compared with the operation of the processor at the highest nominal settings.

The main contributions of our work are the following:

- We follow a modeling approach to examine the effectiveness of VFS, undervolting and overclocking, capturing the trade-off between energy consumption and potential SLA violations.
- We evaluate our approach using realistic node profiles, VM pricing and energy price.

3.1 Models

In this section, we introduce the models that estimate the operational cost due to energy cost and potential SLA violation penalties for the VFS, undervolting and overclocking approaches.

3.1.1 Assumptions

For this work, we consider the following assumptions:

- Energy price is constant and it does not vary in time.
- We consider homogeneous nodes in terms of resources and inherent resilience.
- Each task has an *a priori* known lifetime. Each error manifests as a task failure/crash, and in that case, the entire task is re-executed at the highest nominal operating (V_{max} , f_{max}) point.
- The agreed deadline specified in the SLA concerns the execution of the task at the highest nominal operating point (V_{max} , f_{max}) plus some slack.

Table 3.1 summarizes the variables we use in the models.

Table 3.1: Summary of variables.

Variable	Description
d	Failure sensitivity to frequency scaling (within nominal range).
$delay$	Task execution delay over contractual (SLA) agreement.
C	Average switching capacitance of the processor.
E_{max}	Energy consumption at the highest nominal operating point.
E_e	Expected energy consumption when considering potential failures.
E_x	Energy consumption at a (V_x , f_x) operating point.
f_{max}	Frequency of the processor at the highest nominal operating point.
f_{min}	Frequency of the processor at the lowest nominal operating point.
f_x	Frequency of the processor.
f_{PoFF}	Frequency of the processor at the Point of First Failure.
ins_{num}	Number of instructions of the task.

$intensity$	Intensity of SLA violation penalty wrt. to user paid cost for the service.
$MTTF$	Mean time to failure.
P_{static}	Processor static power.
p_{fail}	Failure probability.
$price_{energy}$	Energy price.
$price_{vm}$	Price of VM per time unit.
$rW_{fraction}$	Fraction of the task that was executed before the failure.
t_{max}	Task execution time at the highest nominal operating point.
t_e	Expected task execution time when considering potential failures.
t_x	Task execution time at (V_x, f_x) .
V_{max}	Supply voltage of the processor at the highest nominal operating point.
V_{PoFF}	Supply voltage of the processor at the Point of First Failure.
V_{th}	Threshold voltage of the processor.
V_x	Supply voltage of the processor.
α	It quantifies part sensitivity to undervolting/overclocking (linear part).
β	It controls the relation between nominal voltage and the respective frequency.
γ	It quantifies part sensitivity to undervolting/overclocking (exponential part).
λ	Failure rate.
λ_0	Failure rate at the highest nominal operating point.

3.1.2 SLA violation penalty

Our QoS metric is the deadline of the task. In order to estimate the penalty due to violation of the deadline, we adopt a model [2] which linearly correlates SLA violation penalty with the extent of delay, after the deadline. Equation 3.1 calculates the penalty of this violation, where $delay$ is the additional time required for completing the task, $price_{vm}$ is the price of the VM per time unit paid by the customer for and $intensity$ is a constant specified in the SLA which connects the compensation penalty to the price paid by the customer for the service.

$$penalty = (delay * price_{vm}) / intensity \quad (3.1)$$

3.1.3 Error Modeling

Equation 3.2 estimates the probability of failure, when processor operates at reduced voltage/frequency margins, and it follows an exponential distribution [1]. t_x is the lifetime of the task and $MTTF$ is the mean time to failure. In order to estimate $MTTF$, we first calculate λ , which is the inverse of $MTTF$ ($MTTF = \frac{1}{\lambda}$). Moreover, the calculation of λ differs among our approaches, hence in the following sections, we discuss this calculation in detail for our three approaches.

$$p_{fail} = 1 - e^{-t_x/MTTF} \quad (3.2)$$

We also estimate the fraction of the task that was executed before a failure according to the following equation:

$$rw_{fraction} = \frac{MTTF}{t_x} + \frac{1}{1 - e^{t_x/MTTF}} \quad (3.3)$$

3.1.4 Energy Consumption Model

We estimate the energy consumption using equation 3.4, where C is the average switching capacitance of the processor and V_x and f_x the voltage and the frequency, respectively, of the operating point. This first term ($C * V_x^2 * f_x$) addresses the calculation of the dynamic power of the processor, while P_{static} corresponds to its static power. Finally, t_x is the execution time of the task at f_x , assuming that no failures occurred. The execution time can be estimated by equation 3.5, where ins_{num} is the number of the instructions executed by the task. At this stage, we assume that the Cycles Per instruction (CPI) do not change significantly with clock frequency, which can be expected to be the case unless the task is memory intensive with bad cache locality.

$$E_x = (C * V_x^2 * f_x + P_{static})t_x \quad (3.4)$$

$$t_x = (ins_{num} * CPI)/f_x \quad (3.5)$$

We use equation 3.6 to estimate the expected energy consumption for the execution of a task at (V_x, f_x) operating point. The first term $((1 - p_{fail}) * E_x)$ calculates the contribution to the expected energy E_e , if there are not any failures. The second term $(p_{fail} * (rw_{fraction} * E_x +$

E_{max})) addresses the contribution to E_e , when a failure occurs, and namely, $rw_{fraction}$ shows the fraction of the task that was executed before the failure, operating at (V_x, f_x) and consuming E_x . Due to this failure the entire task has to be re-executed at (V_{max}, f_{max}) consuming E_{max} .

$$E_e = (1 - p_{fail}) * E_x + p_{fail} * (rw_{fraction} * E_x + E_{max}), rw_{fraction} < 1 \quad (3.6)$$

Finally, we estimate the energy cost using equation 3.7, where $price_{energy}$ is the price of the energy.

$$cost_{energy} = E_e * price_{energy} \quad (3.7)$$

3.1.5 Energy Reduction Methodologies

In this section, we discuss the estimation of the failure rate (λ) and the expected execution time (t_e) for each one of the three approaches.

Voltage and Frequency Scaling (VFS model)

In VFS, we concurrently manipulate the voltage and the frequency of the processor for operation to a different nominal operating point (V_x, f_x) than the max (V_{max}, f_{max}) . We calculate the corresponding frequency (f_x) for voltage (V_x) using equation 3.8 [5], where β is a CPU model-specific constant and V_{th} is the threshold voltage of the processor.

$$f_x = \beta * (V_x - V_{th})^2 / V_x \quad (3.8)$$

Equation 3.9 estimates the failure rate at (V_x, f_x) , where λ_0 is the failure rate at the maximum nominal operating point, d reflects the hardware proneness to failure when scaling voltage and frequency, and f_{max} and f_{min} are the highest and lowest nominal frequencies, respectively. In VFS, error rates are correlated to the operating voltage and frequency, as in lower power/performance operating points circuits are more prone to transient faults due to lower-energy upsets (such as cosmic particles).

$$\lambda = \lambda_0 * e^{\frac{d * (f_{max} - \beta * (V_x - 2 * V_{th} + \frac{V_{th}^2}{V_x}))}{f_{max} - f_{min}}} \quad (3.9)$$

In this approach, the frequency of the processor changes, hence the performance and the execution time of the task are affected directly. We estimate the expected execution time of the task using equation 3.10. If a failure did not occur, the first term (t_x) provides the execution time of the task. However, if a failure occurred the first two terms corresponds to the execution time before the failure, while the third one estimates the time that is required for re-execution of the task at the highest nominal operating point (V_{max}, f_{max}).

$$t_e = t_x - p_{fail} * (1 - rw_{fraction}) * t_x + p_{fail} * t_{max}, rw_{fraction} < 1 \quad (3.10)$$

Voltage Scaling (VS model)

In this approach, we keep the frequency of the processor at the maximum level (f_{max}), and we reduce the voltage supply (V_x). We use equation 3.11 [4] to estimate the failure rate, where V_{PoFF} is the voltage corresponding to the Point of First Failure ($PoFF$) [6] and α and γ are constants we obtain by the data provided in [14] and [6]. $PoFF$ corresponds to the point at which the error rate equals 1 error every 10 million cycles and beyond $PoFF$, error rate increases exponentially with the decrease of voltage [6], [14]. Timing errors are the dominant cause of failures when undervolting.

$$\lambda = \alpha * 10^{\gamma * (V_x - V_{PoFF})} \quad (3.11)$$

As the frequency of the processor does not change, the execution time of the task remains the same (t_{max}), and equation 3.12 estimates the expected execution time, when we only change the voltage of the processor, keeping the frequency at f_{max} . The first term (t_{max}) corresponds to the time required for executing the task, while the second term ($p_{fail} * rw_{fraction} * t_{max}$) addresses the execution time before a failure occurs. Note that only the first term contributes to t_e , if a violation does not occur.

$$t_e = t_{max} + p_{fail} * rw_{fraction} * t_{max}, rw_{fraction} < 1 \quad (3.12)$$

Frequency Scaling (FS model)

FS and VS models are similar. However, in FS model, we keep the voltage at maximum level V_{max} , and we increase the frequency of the processor. Moreover, the failure rate is

estimated by equation (3.13) using the frequency rather than the voltage of the processor, where f_{PoFF} is the frequency that corresponds to the Point of First Failure.

$$\lambda = \alpha * 10^{\gamma * (f_{PoFF} - f_x)} \quad (3.13)$$

The expected execution time of the task is estimated similar to VFS technique by equation 3.10, as in this case the frequency changes too.

3.1.6 Total cost

Finally, we estimate the total operating cost using equation 3.14, which consists of the energy cost ($cost_{energy}$) and potential SLA violation penalties ($penalty$).

$$cost = cost_{energy} + penalty \quad (3.14)$$

3.2 Evaluation

We use the characteristics of the Intel Core i7 4820K to evaluate our models. This processor operates at 3.7 GHz with 1.06 V (highest nominal operating point). We also use realistic values for the price of VMs [17] and the energy price [3]. Namely, the prices are 0.226 \$/hour (compute optimized VM) and 0.07 \$/KWh for the VM and the energy, respectively. We experiment with different task duration times (1 second, 10 seconds and 100 seconds) and different values for *intensity* (1, 2 and 4). Moreover, we assume that each VM contains one task, and finally, we quantify the effects of protection mechanisms [4] which manage to identify and mask a percentage of sporadic faults.

We present our results in Figures 3.1, 3.2 and 3.3, and for each approach, we illustrate three different cases. The leftmost figures present the energy cost reduction, for tasks with different execution times (1 second, 10 seconds and 100 seconds). The figures in the middle quantify the operating cost reduction due to energy consumption and potential SLA violation penalties. For this case, we use tasks with an execution time of 100 seconds and various values for *intensity* (1, 2 and 4). Finally, the rightmost figures illustrate the effect of fault protection/masking mechanisms to operating cost. Again, we assume tasks of execution time of 100 seconds, but we fix *intensity* to 4. The baseline operating cost for all cases is the cost for executing the tasks at the highest nominal operating point (V_{max}, f_{max}).

We observe that the most efficient approach is voltage scaling, which manages to reduce the operating cost (13.31 % on average for the different scenarios depicted in Figure 3.1). Frequency scaling is the second most efficient approach, which manages to reduce the execution time of the tasks, however it increases slightly the power consumption of the processor. Although frequency scaling is not very effective at reducing the operating cost, tasks finish their execution faster, thus the system is available for new tasks earlier, which can increase the profit for the operator. Even in the absence of additional jobs, the cost can also be reduced by switching nodes to lower P- and C-states or powering-off nodes, reducing the energy cost. Finally, the Voltage and Frequency Scaling approach is the least effective one, as it only exploits the slack over t_{max} to identify a nominal point with a lower energy footprint (compute intensive tasks). These benefits are marginal, and when the slack is consumed, SLA violation penalties outweigh any energy cost reduction.

One more interesting observation for VS and FS models, is that the system starts to operate at a non cost-effective configuration as we closely approach V_{PoFF} and f_{PoFF} due to the increased number of errors. The corresponding values for V_{PoFF} and f_{PoFF} are 0.901 V and 4.2 GHz, respectively. When errors start to occur on the system, the energy consumption increases due to re-execution at (V_{max}, f_{max}) for the failed tasks, and the cost increases even more, as we approach the $PoFF$. However, we observe that, after a point, the cost stabilizes because the tasks fail almost immediately due to the high error rate. The operating cost after this point equals the cost of the re-execution of the task at (V_{max}, f_{max}) (should the system overall survive the faults). Therefore, the target of VS and FS is to approach $PoFF$, avoiding to reach it, in order to achieve the highest cost gains. In our experiments the leftmost figures indicate that the optimal ranges for VS and FS approaches are 0.947–0.967 V and 3.98–4.04 GHz, respectively. These ranges depend on the execution time of tasks, and lower execution times allow for more aggressive configurations. On the other hand, the optimal configuration for the VFS approach depends on the available slack rather than the execution time of tasks.

Moving to the middle set of figures, we observe the contribution of potential SLA violation penalties to the operating cost. We assume a slack of 10 % over the expected execution time at the highest nominal operating point (t_{max}) for the deadline of the task. We experiment with different values for *intensity* (the effect of *intensity* is more prominent at lower values of slack), where higher values of *intensity* are more beneficial for the provider, as they limit the penalty. In middle figures, we also observe that the effect of SLA violation

penalties is negligible, when we operate the processor at the optimal ranges of voltage and frequency. However, moving closely to $PoFF$, where the errors increase exponentially, the SLA violation penalty corresponds to a significant fraction of the operating cost. On the other hand, for the VFS approach, where we manipulate the voltage and frequency concurrently, the SLA violation penalties occur even with a small decrease of V_x and f_x , namely, after the point where the slack is fully depleted due to lower performance.

Finally, we study cost reduction when we apply protection mechanisms which can detect and mask the effects of a percentage of errors. According to [4], such mechanisms may mask up to 70 % of errors. As we observe, there is a limited improvement in the optimal operating points for VS and FS, which however does not translate to a measurable reduction of cost. However, those mechanisms are valuable in a different way: they provide a margin where errors may be detectable, yet not catastrophic for the system. In other words, they can provide observability to the fact that the system approaches the problematic range. For the VFS model, failure rate for nominal operating points is already very low, therefore protection mechanisms do not have any measurable impact.

3.3 Related Work

[13] indicates that Dynamic voltage and frequency scaling (DVFS) in cloud deployments has the potential to reduce power consumption and increase profit. DVFS is also used by [9] and [10] to reduce the energy footprint of cloud applications and infrastructure. The work in [10] can, in addition, predict potential errors. [7, 12] exploit Dynamic Voltage Scaling to minimize energy consumption for HPC systems. [15] applies frequency scaling in order to minimize the energy consumption taking into account potential penalties. The authors in [16] study the behavior of an ARMv8 64-bit CPU when they undervolt it, providing a detailed characterization and prediction models in order to achieve energy efficiency and preserve operation correctness. The studies in [2, 11] introduce models that involve several QoS metrics used in the SLAs between the consumers and providers, through a complete environment for cloud applications. The resulting models focus on maximizing the provider profit, while taking into consideration SLA violation penalties.

In our work, we study three approaches: the more conservative technique of VFS and the two more aggressive techniques of voltage and frequency overscaling. Although these

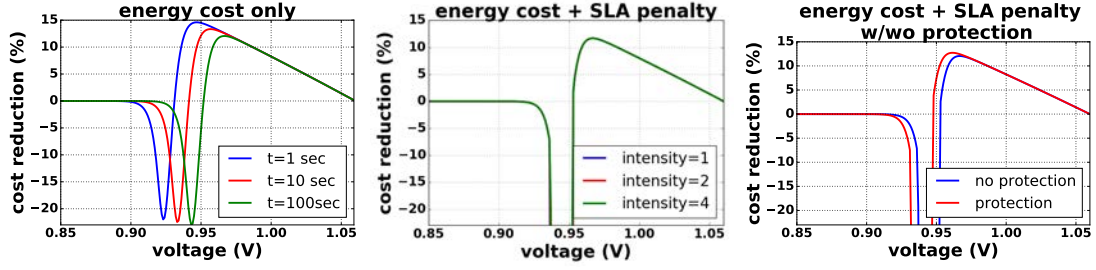


Figure 3.1: Operator cost reduction using Voltage Scaling.

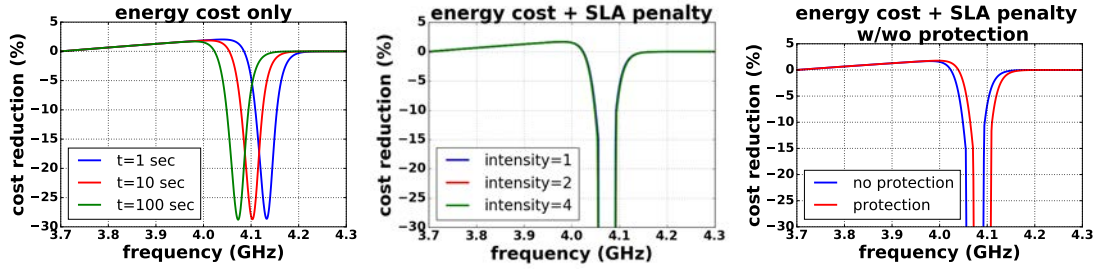


Figure 3.2: Operator cost reduction using Frequency Scaling.

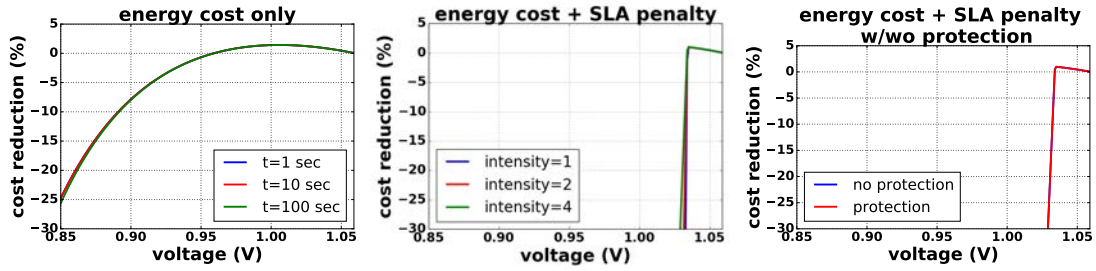


Figure 3.3: Operator cost reduction using Voltage and Frequency Scaling.

techniques can increase the probability of errors, our work shows that there is a range of operating configurations which can minimize the cost for the cloud/edge provider, even when potential failures are considered. Eventually, due to timing errors, we take into account the potential SLA violation penalties that may reduce the profit of energy saving and identify the optimal operating point (trade-off between energy reduction and QoS) that minimizes the cost.

Chapter 4

Exploitation of CPU Voltage Margins for Cost Reduction at Cloud Infrastructures

In the previous chapter, we focused on a theoretical analysis to identify whether CPU configurations at reduced margins could result in improved energy efficiency of compute nodes. In this chapter, we employ a simulation environment, and use realistic parameters attained from real server nodes, to investigate if the configuration at reduced CPU voltage margins is cost-efficient. More specifically, we focus on the question "Can operation at reduced CPU voltage margins increase the profit of Cloud infrastructure providers?". To this end, we study the cost reduction for cloud infrastructures combining the traditional techniques of VM consolidation and VFS with the exploitation of CPU voltage margins, examining the trade-off between the energy cost and potential SLA violation penalties. The main contributions of our work are the following:

- We introduce an analytic and architecture-agnostic model that captures the trade-off between the energy cost and the risk of SLA violation penalties.
- We introduce a novel scheduling and system configuration policy for cloud datacentres that exploits configuration at reduced margins combined with VM consolidation and VFS. The policy considers the cost of potential SLA violation penalties due to failures, node overloading and VM restarts.
- We consider two different types of SLA models to quantify the cost of violation penalties due to failures, node overloading and VM restart.
- We evaluate the proposed policy via simulation experiments for the two types of the

SLA models and varying/constant energy prices, using realistic node profiles that are derived from real systems, and show that our policy achieves significantly better results compared with a state-of-the-art policy.

For the experimental evaluation, we use the parameters of two commercial server-class systems with different processor architectures, namely, an Intel Xeon E3-1220 v5 and an ARM-based Ampere Computing X-Gene 3. We experimentally quantify the voltage margins, the power consumption and the computational capacity of those systems. We also consider that a typical datacentre runs jobs with various characteristics and different behavior when applying frequency scaling. Finally, we evaluate our approach against a state-of-the-art policy, which we outperform in terms of cost by up to 17.24%.

4.1 Model

We introduce an analytic and architecture agnostic model which captures the most important server configuration and VM scheduling aspects relevant to configuration at reduced voltage margins. We assume that nodes are homogeneous in terms of CPU hardware. Table 4.1 summarizes the variables we use in the models.

Table 4.1: Summary of variables.

Variable	Description
$CPU_{m,f_i,k}^{alloc}$	The allocated CPU capacity of VM_m at f_i for the k^{th} timeslot.
CPU_{m,f_i}^{req}	The requested CPU capacity of VM_m at f_i .
$CPU_{m,f_{max}}^{req}$	The requested CPU capacity of VM_m at f_{max} .
CPU_{m,f_i}^{SLA}	The requested CPU capacity of VM_m defined in SLA for f_i .
$CPU_{m,f_{max}}^{SLA}$	The requested CPU capacity of VM_m defined in SLA for f_{max} .
$Cost_i$	The cost of n_i .
$Energy_i$	The energy cost of n_i .
f_i	The frequency of the processor.
f_{max}	The frequency of the processor at the highest performance nominal operating point.
k	The k^{th} timeslot.
K	The number of timeslots of a scheduling period.

$Load_{i,k}$	The load of n_i node at k^{th} timeslot.
M	The number of the VMs.
MEM_i	The memory amount of n_i .
MEM_m^{SLA}	The requested memory of VM_m defined in SLA.
N	The number of nodes.
n_i	The i^{th} node.
$Pfail_{V_i,f_i}$	The probability of failure at reduced voltage margins configuration.
p	The scheduling period.
$price_m$	The VM_m price per time unit.
q_m	A multiplier that increases the severity of the SLA penalty for VM_m .
Res_m	The number of timeslots required for VM_m for rescheduling and restart.
$slotT$	The duration of each timeslot.
$SLAV_{i,m,k}$	The SLA violation cost for a timeslot for VM_m on n_i .
$SLAV_i^{nofail}$	The SLA violation cost for an entire period for n_i without a failure.
$SLAV_i^{fail}$	The SLA violation cost for an entire period for n_i for a failure.
V_{max}	The supply voltage of the processor at the highest performance nominal operating point.
V_i	The supply voltage of the processor.
VM_m	The m^{th} VM.
$u_{i,k}$	The utilization of n_i at k^{th} timeslot.
$unitPrice$	The energy price per time unit.

4.1.1 Nodes and CPU operating points

We assume a datacentre which consists of N nodes n_i , $1 \leq i \leq N$, each one with finite memory Mem_i . There are different nominal operating points (V, f) (V is the supply voltage and f is the frequency) for the CPU of the nodes, and (V_{max}, f_{max}) represents the highest performance / highest power nominal operating point.

For each of the nominal operating points, the CPU can operate at a sub-nominal and more energy-efficient point (V_x, f) where $V_x < V$. However, execution at reduced voltage margins can lead to a failure with probability $Pfail_{V_x,f}$, even though we carefully select a safe (V_x, f) pair. We assume that execution at a nominal operating point has a negligible (zero) probability

of a failure. We determine the reduced margins points and the respective failure probabilities by experimental characterization on real hardware (see Section 4.3).

4.1.2 Application workload – VMs

There are M different application VMs VM_m , $1 \leq m \leq M$, that are submitted for execution on the server farm. These VMs request a different amount of the resources of the node each. The respective maximum requirements are defined in a service level agreement (SLA). Mem_m^{SLA} and $CPU_{m,f_{max}}^{SLA}$ are the amount of memory and the upper bound (maximum) for the CPU computational capacity requested by VM_m , respectively, specified in the SLA. We assume that the requested memory is allocated in full before a VM starts running, and remains allocated during the entire lifetime of the VM. The memory serves as a proxy for any static resource requirement.

$CPU_{m,f_{max}}^{SLA}$ is expressed as the fraction of the full CPU computational capacity at the maximum frequency f_{max} . In contrast with the memory requirements, the required CPU capacity is dynamic and the VM does not require the maximum CPU computational capacity $CPU_{m,f_{max}}^{req} \leq CPU_{m,f_{max}}^{SLA}$ at all times. Moreover, the computational capacity $CPU_{m,f}^{req}$ of VM_m depends on the frequency of the CPU, $CPU_{m,f}^{req} \geq CPU_{m,f_{max}}^{req}$. When the CPU frequency drops below f_{max} , the effective total CPU capacity is reduced, hence the relative CPU capacity (percentage of total) that is required by a VM to achieve the same performance effectively increases. Moreover, the amount of this increase depends on the sensitivity of the VM to frequency scaling. Compute-intensive applications are more sensitive when we scale the frequency than memory- or I/O-intensive ones. We determine the performance sensitivity to frequency scaling for different application classes via experimental characterization on real hardware (see Section 4.3).

4.1.3 VM scheduling

Scheduling is done at the granularity of periods, and each period has fixed length equal to K timeslots of $slotT$ time units each. For symmetry, we assume that new VMs arrive and existing VMs finish at the boundaries of scheduling periods. After each period, the scheduler determines (a) the mapping of new VMs, as well as potential reschedule/restarts of existing VMs to nodes, and (b) the (V, f) setting of each node, for the next period. Res_m is the number of timeslots needed to reschedule and restart VM_m to node n_i . Without loss of generality,

we assume that VM restarts take at least one timeslot and on average finish fast, i.e., $1 \leq Res_m \ll K$. We assume that restarts do not create significant load on the receiving node, and that the VM remains unavailable during restart.

The settings of each node n_i are represented with (V_i, f_i) (nominal or reduced). Also, we use matrix H to encode the placement of VMs to nodes, where $H[i, m] = 1$, if node n_i hosts VM_m , else $H[i, m] = 0$. In other words, this matrix represents the mapping of VMs to nodes for the respective period. Also, n_i must have enough memory to host the VMs that are mapped to it: $Mem_i \geq \sum_{m=1}^M H[i, m] \times Mem_m^{SLA}$. Beyond H , we also use the 3-dimensional matrix X to model the restart overhead. The last dimension of X corresponds to timeslots within the current period. $X[i, m, k] = 1$ if VM_m runs on n_i in the k th timeslot, otherwise $X[i, m, k] = 0$. Therefore, for a restart overhead of Res_m timeslots, $X[i, m, k]$ will be 0 for $0 \leq k \leq Res_m - 1$ and $X[i, m, k]$ will be 1 for $Res_m \leq k < K$.

4.1.4 CPU load and CPU capacity allocation to VMs

We use the following equation to estimate the load of the VMs that run on n_i for timeslot k :

$$Load_{i,k} = \sum_{m=1}^M X[i, m, k] \times CPU_{m,f_i}^{req} \quad (4.1)$$

Although, we assume that the requirements of the VMs do not vary during a period, the load may change within a period due to the restart of VMs which start running on the node with some delay. The following equation estimates the CPU capacity allocated for VM_m which depends on whether the node is under- or over-loaded:

$$CPU_{m,f_i,k}^{alloc} = \begin{cases} X[i, m, k] \times CPU_{m,f_i}^{req} & \text{if } Load_{i,k} \leq 1 \\ X[i, m, k] \times \frac{CPU_{m,f_i}^{req}}{Load_{i,k}} & \text{if } Load_{i,k} > 1 \end{cases} \quad (4.2)$$

More specifically, VM_m will get the requested CPU capacity if a node is under-loaded during the given timeslot, otherwise the CPU capacity will be shared among VMs proportionally to their requests (we assume equal priorities). The multiplication with $X[i, m, k]$ is so that $CPU_{m,f_i,k}^{alloc} = 0$ in case VM_m is under restart during the given timeslot.

4.1.5 Failures due to operation at reduced margins

Configurations at reduced voltage margins can lead to a failure with probability $P_{fail_{V_i,f_i}}$.

We adopt a simple model, where: a) every failure leads to a node crash, affecting all VMs

that are hosted on the node, and b) failures occur at the beginning of the period, and the failed node remains unavailable during the entire period. Moreover, the VMs hosted at a failed node are not re-scheduled to another node during the period when the failure occurs, so they do not run for this period. Also, if a failure occurs on node n_i , the node recovers and it is able to host VMs by the next period, which is a realistic assumption for a scheduling period that is relatively large.

4.1.6 SLA violation cost

A VM may not always get the requested CPU capacity at every timeslot. In other words, there may be some timeslots k where $CPU_{m,f_i,k}^{alloc} < CPU_{m,f_i,k}^{req}$. This can happen because the VM is restarting, because the host node is overloaded or because the host node has suffered a failure. In all these cases, the provider pays an SLA violation penalty to the VM owner. We study two different models, in the spirit of [11], as follows.

Percentage price refund (PPR) SLA model: The penalty that the provider pays to the customer depends on the percentage of the CPU capacity that is not allocated for VM_m and the duration of the violation:

$$SLAV_{i,m,k} = q_m \times \frac{CPU_{m,f_i}^{SLA} - CPU_{m,f_i,k}^{alloc}}{CPU_{m,f_i}^{SLA}} \times SlotT \times price_m \quad (4.3)$$

More specifically, we assume that the penalty for timeslot k is proportional to the *extent* of the violation during that timeslot, expressed as the difference between the provisioned CPU resources, adjusted to the computational capacity of the CPU at frequency f_i CPU_{m,f_i}^{SLA} , and the CPU resources that were actually allocated to the VM $CPU_{m,f_i,k}^{alloc}$, divided by CPU_{m,f_i}^{SLA} .

Full price refund (FPR) SLA model: In case of the slightest violation, the penalty is computed as if no CPU capacity were allocated to VM_m at all. Therefore, the penalty equals to:

$$SLAV_{i,m,k} = q_m \times SlotT \times price_m \quad (4.4)$$

Note that this is equivalent to Equation 4.3 for $CPU_{m,f_i,k}^{alloc} = 0$. Obviously, the FRP model is stricter compared to PPR, making node overloads (due to VM consolidation) significantly more costly.

In both cases, $price_m$ is the price charged to the client for executing VM_m for a time unit

as agreed in the SLA, $SlotT$ is the length of a timeslot in time units, and q_m is a multiplier that increases the severity of the SLA penalty according to the importance of VM_m .

So, the SLA violation cost over an entire period for all VMs running on node n_i that does not fail, can be expressed as

$$SLAV_i^{nofail} = \sum_{k=1}^K \sum_{m=1}^M X[i, m, k] \times SLAV_{i,m,k} \quad (4.5)$$

However, if n_i fails, none of the VMs hosted on this node will get any CPU capacity for this period, hence we use the following equation to estimate the penalty:

$$SLAV_i^{fail} = K \times SlotT \times \sum_{m=1}^M H[i, m] \times q_m \times price_m \quad (4.6)$$

4.1.7 Energy cost

The CPU utilization at timeslot k for node i (n_i) is given by

$$u_{i,k} = \min\{Load_{i,k}, 1\} \quad (4.7)$$

Let $Power(V_i, f_i, u_{i,k})$ be the power consumed by n_i at operating point (V_i, f_i) (nominal or reduced), when the CPU utilization is $u_{i,k}$. We determine this function via experimental characterization of real hardware (see Section 4.3). The total energy cost for node n_i over an entire period is estimated as follows (assuming a fixed unit price $unitPrice$ for energy)

$$Energy_i = unitPrice \times \sum_{k=1}^K Power(V_i, f_i, u_{i,k}) \times SlotT \quad (4.8)$$

Note that if n_i has not been assigned any VMs, it will remain idle and its utilization $u_{i,k}$ will be 0. Moreover, we assume that failed (crashed) nodes do not consume any energy.

4.1.8 Overall operating cost estimation

Equation 4.9 estimates the cost for operating node n_i at (V_i, f_i) . This equation estimates the cost considering the probability of a node failure due to the configuration of n_i . More specifically, if n_i fails the provider will pay the penalty for the VMs that are hosted on this node for the entire scheduling period ($Pfail_{V_i, f_i} \times SLAV_i^{fail}$). If the node does not fail, the cost consists of the energy cost plus potential SLA violation penalties due to node overloading $((1 - Pfail_{V_i, f_i}) \times (SLAV_i^{nofail} + Energy_i))$. This equation is an important tool when

looking for the optimal operating point of each node for the next period. It considers the energy consumption, as well as the cost of any potential failures or overloads of the node.

$$Cost_i = Pfail_{V_i, f_i} \times SLAV_i^{fail} + (1 - Pfail_{V_i, f_i}) \times (SLAV_i^{nofail} + Energy_i) \quad (4.9)$$

Finally, the total estimated cost for the next period over all nodes is $\sum_{i=1}^N Cost_i$. Our objective is to devise policies that consider this cost in order to re-configure nodes and schedule the VMs on nodes in an educated manner, by exploiting the reduced CPU voltage margins to save energy as long as this outweighs the potential cost due to the increased probability of SLA violations as a result of failures, node overloads and VM restarts.

4.2 VM Scheduling and System Configuration Policy

We introduce *XM-VFS*, a policy that considers the trade-off between energy cost and potential SLA violation penalties to minimize the cost of cloud infrastructure providers. *XM-VFS*: a) maps the VMs to nodes using a set of heuristics, and b) selects the configuration (V_i, f_i) for each node n_i that minimizes the total cost. An important novelty of the policy is that it exploits operation at reduced voltage margins on top of the traditional arsenal of VFS and VM consolidation.

VM scheduling and node configuration decisions take place at the beginning of each period, hence we execute *XM-VFS* at scheduling period boundaries. It consists of 4 steps: a) detection of overloaded nodes and selection of candidate VMs to be restarted to other nodes, b) mapping of the restarted VMs and any newly arriving VMs to their new hosting nodes, c) powering-off underutilized nodes if their VMs can be restarted to other nodes, and d) re-configuration of the operating point (V_i, f_i) of n_i if needed. The first 3 steps are similar in nature to previous work on VM consolidation [19]. Actually we use the same heuristics as [19] for step (c). In the following discussion we focus more on steps (a), (b) and (d).

In order to identify potential node overloads and decide the VM placement and node configuration for the next period p , we first determine the most power efficient operating point $((V, f)_{i,p-1}^{efficient})$ for the workload of the previous period $p - 1$ for each n_i . This step considers both node and VM characteristics. Then, *XM-VFS* predicts the resource requirements of each VM_m for the next period p , using historical information for the last 10 periods with exponential aging. Given this prediction we estimate the required computational capacity that

corresponds to frequency $f_{i,p-1}^{efficient}$ for each node n_i . Algorithm 1 describes this process in lines 1-2. Afterwards, we use a static utilization threshold ($MaxUtil$), which is the same for all nodes, to identify potential node overloads. For the nodes that are expected to be overloaded during the next period p , we re-schedule and restart to other nodes the lowest number of VMs necessary to lower the expected load of n_i below $MaxUtil$. At this step, *XM-VFS* selects the least costly VMs for restart. Note that restart cost is a factor of the restart overhead Res_m , $price_m$ and q_m (lines 4-6). *XM-VFS* performs this step at the beginning of each scheduling period, independently and concurrently on each node.

Algorithm 1 Node overload detection and VM selection for migration

Input: $n_i, vmList_{n_i}$
Output: VMsForScheduling

```

1:  $f_{i,p} \leftarrow f_{i,p-1}^{efficient}$ 
2: estimate CPU utilization  $u_{i,f_{i,p}}^p$ 
3: if  $u_{i,f_{i,p}}^p > MaxUtil$  then
4:   for  $VM_m$  in  $vmList_{n_i}$  do
5:      $Cost_m^{res} \leftarrow Res_m \times SlotT \times q_m \times Price_m$ 
6:   end for
7:   while  $u_{i,f_{i,p}}^p > MaxUtil$  do
8:      $Cost_{min} \leftarrow \infty$ 
9:     for  $VM_m$  in  $vmList_{n_i}$  do
10:      if  $Cost_m^{res} < Cost_{min}$  then
11:         $Cost_{min} \leftarrow Cost_m^{res}$ 
12:         $VMMinCost \leftarrow VM_m$ 
13:      end if
14:    end for
15:     $vmList_{n_i}.remove(VMMinCost)$ 
16:     $VMsForScheduling.add(VMMinCost)$ 
17:    re-evaluate  $u_{i,f_{i,p}}^p$ 
18:  end while
19: end if

```

At step (b), we map VMs that require re-scheduling, and newly arriving VMs to nodes. This is essentially a bin packing problem, with VMs corresponding to items and nodes cor-

responding to bins. We use the Best Fit Decreasing (BFD) algorithm to allocate the VMs to nodes. At this step, *XM-VFS* respects: a) the $(V, f)_{i,p-1}^{efficient}$ of each n_i , and b) $MaxUtil$. In other words, n_i can host VM_m , if the sum of the load of VM_m and the load of n_i does not exceed $MaxUtil$ for $(V, f)_{i,p-1}^{efficient}$. This step has linearithmic ($\mathcal{O}(m \log m)$) complexity w.r.t the total number of VMs to be mapped on the nodes.

At this point, *XM-VFS* uses the heuristic in [19] to identify opportunities to power-off nodes. For all nodes, starting from the least utilized one, it is evaluated whether all hosted VMs can be restarted to other nodes without overloading them. Potential destination nodes are identified using again the BFD heuristic. Assuming that the number of VMs per node is relatively small, the complexity of this step is $\mathcal{O}(n \log n)$, where n is the number of nodes in the datacentre.

Algorithm 2 Node configuration for cost minimization

Input: n_i , *OperatingPointList*

Output: Node n_i configuration $(V, f)_{i,p}^{efficient}$

```

1:  $Cost_{min} \leftarrow \infty$ 
2:  $(V, f)_{i,p}^{efficient} \leftarrow (V, f)_i^{max,nominal}$ 
3: for  $(V, f)_{i,p}$  in OperatingPointList do
4:   estimate  $Cost_i$  at  $(V, f)_{i,p}$ 
5:   if  $Cost_{min} > Cost_i$  then
6:      $Cost_{min} \leftarrow Cost_i$ 
7:      $(V, f)_{i,p}^{efficient} \leftarrow (V, f)_{i,p}$ 
8:   end if
9: end for
10: configure node  $n_i$  at  $(V, f)_{i,p}^{efficient}$ 

```

Finally, *XM-VFS* decides the optimal (V_i, f_i) configuration of each node, given the VM placement for the next scheduling period p . This is described by Algorithm 2. The optimal node configuration minimizes overall cost at the node level, therefore for a fixed VM-to-nodes mapping the total cost for the entire datacentre as well, taking into account both energy cost and potential SLA violation costs. We use Equation 4.9 in order to estimate the cost for each operating point (nominal and reduced margins) and we choose the one that introduces the lowest cost. Algorithm 2 is executed independently on each node and its complexity is

Table 4.2: Target hardware platforms.

Parameter	Intel platform	ARM platform
Architecture	x86-64 (Skylake)	ARMv8 (Ampere X-Gene 3)
# of Cores	4	32
TDP	80 W	125 W
Max Frequency	3.3 GHz	3.0 GHz
Memory	32 GB	128 GB

$\mathcal{O}(1)$ with respect to the number of VMs and nodes.

4.3 Hardware Characterization

For our simulation-based evaluation we use parameters from real systems which are equipped with commercially available CPUs. Table 4.2 summarizes the systems used for this characterization. Following, we explain how we determine the parameters that capture the behavior of a node in the model described in Section 4.1 and, specifically, (i) the reduced voltage margins operating points, (ii) the failure probability when operating at such points, (iii) the power estimation function for different operating points and utilization levels, and (iv) the impact of frequency scaling on the performance of applications.

4.3.1 Voltage margin characterization

The Skylake processor uses the P-State manager / DVFS governor to dynamically control the operating point (V, f) . For our evaluation, we pick four frequency points, which cover the range of the most common frequencies applied in the *balanced* mode of the governor, namely 3.3, 3.0, 2.5 and 2.0 GHz. The average nominal supply voltages for these frequencies are 1147, 1075, 922, and 850 mV, respectively. On the other hand, the X-Gene 3 processor, targeting micro-servers, does not support DVFS. It allows only frequency scaling in nominal operation. We choose four frequency points, at 3.0, 2.2 1.3 and 0.4 GHz, covering the entire range that is supported by this CPU. In all cases, the nominal supply voltage equals 880 mV.

For each nominal operating point (V, f) , we experimentally determine a corresponding

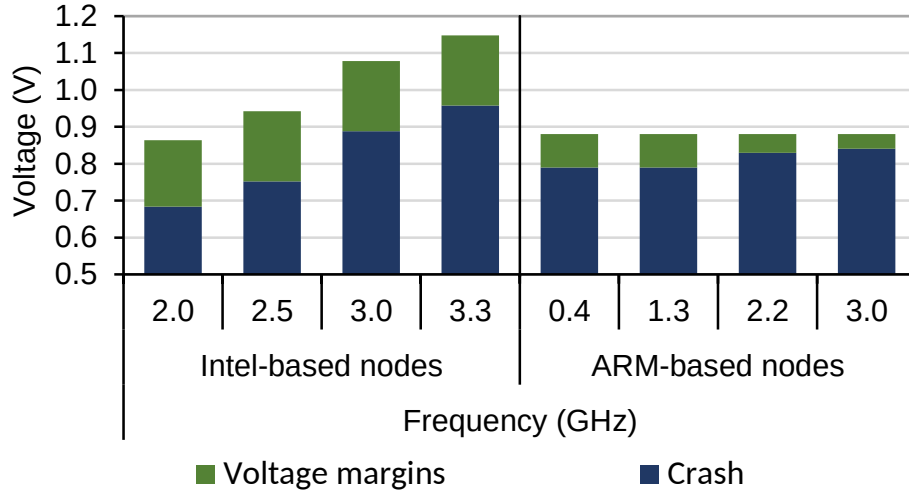


Figure 4.1: Voltage margins characterization for the Skylake (*left*) and the X-Gene 3 CPUs (*right*).

reduced voltage margins operating point (V_x, f) . To quantify these points, we perform experiments with 24 benchmark applications from the SPECCPU 2006 suite [20] stressing various CPU components. An experiment consists of running each application for a continuous period of 10 minutes (if needed, we run the application several consecutive times). In our experiments, we fix the CPU frequency to f , and gradually reduce the CPU voltage in steps of 10 mV, starting from the nominal voltage V . We stop when one of the experiments leads to any type of abnormal behavior (failure), and identify the immediately preceding voltage level (that did not result into a failure) as V_x for the frequency f .

Figure 4.1 illustrates the degree of undervolting that can be applied to each nominal operating point of the Skylake and X-Gene 3 CPUs. The top (green) area of the bars denotes the range of sub-nominal voltages that did not result in any failure. The lower (blue) area of the bars are the voltages that lead to failures for some or all applications. For the reduced voltage margins operating point at each frequency we choose the lowest voltage within the green area. In other words, we choose the lowest sub-nominal voltage that did not lead to any failure. For the Skylake CPU, the sub-nominal voltages for the four frequency points of 3.3, 3.0, 2.5 and 2.0 GHz are 929, 865, 741 and 666 mV, respectively. For the X-Gene 3 CPU, the sub-nominal voltages for the four frequency points of 3.0, 2.2 1.3 and 0.4 GHz are 840, 830 790 and 790 mV, respectively. The characterization process took about 32 hours for both machines. While the duration of the characterization is not negligible, this process needs to be performed once, when adding a new node to the datacentre, and then only very sporadically

to capture aging effects.

4.3.2 Failure probability

Hardware failures are extremely rare [22] at nominal (V, f) points. Failures are expected to occur more frequently, if the CPU operates at a non-nominal point. We validate the safety of the identified sub-nominal voltages of the above experiments by executing multiple experimental campaigns of random workloads at each reduced operating point for 6 consecutive days each. During these experiments, we observed no errors.

Being unrealistically pessimistic, we assume that the 6 days of execution time without any errors is the mean time to failure (*MTTF*), for all tested reduced margin points which corresponds to a failure rate of one failure every 518,400 seconds. Again, we pessimistically assume that these failures are fatal, ignoring the possibility of correctable errors [21], and we estimate the failure probability $P_{fail_{V_x, f}}$ within a scheduling period for a node that operates at any reduced margins point (V_x, f) using this failure rate. For example, assuming a scheduling period of 300 seconds, if a node is configured to operate at reduced margins, the failure probability is $300/518,400 = 0.000579$. We use this $P_{fail_{V_x, f}}$ in our simulations to show that even with a relatively high failure probability, educated undervolting can provide a significant increase of the profit margin for the infrastructure provider.

4.3.3 Power consumption

We estimate the power consumption function $Power(V, f, u)$, again performing experiments for the two hardware platforms. We run a series of experiments for different utilization levels, varying also the number of CPU cores used for numerous nominal voltage-frequency points. We use a wall power-meter to record the values of power consumption. Then, based on these data, we use linear regression to compute the parameters of the model $Power = A + B * u * V^2 * f$, which estimates the power consumption as a function of CPU utilization u , supply voltage V and frequency f . This model serves as the power estimation function in our simulation experiments. We should also note that we model and report energy consumption gains at the plug, rather than on the processor.

We get $A = 34.01$ and $B = 18.98$ for the Skylake CPU predicting the power consumption with $R^2 = 0.98$ and a root mean square error (RMSE) of 4.40 for the unseen configurations at nominal operating points. For the X-Gene 3 CPU, $A = 52.48$ and $B = 34.38$, with $R^2 = 0.99$

and $RMSE = 2.73$. The low RMSE indicates that the predictions of both models are close to the actual power consumption. We have also performed experiments for configurations at reduced margins to verify the accuracy of the power estimation models.

4.3.4 Performance degradation due to frequency scaling

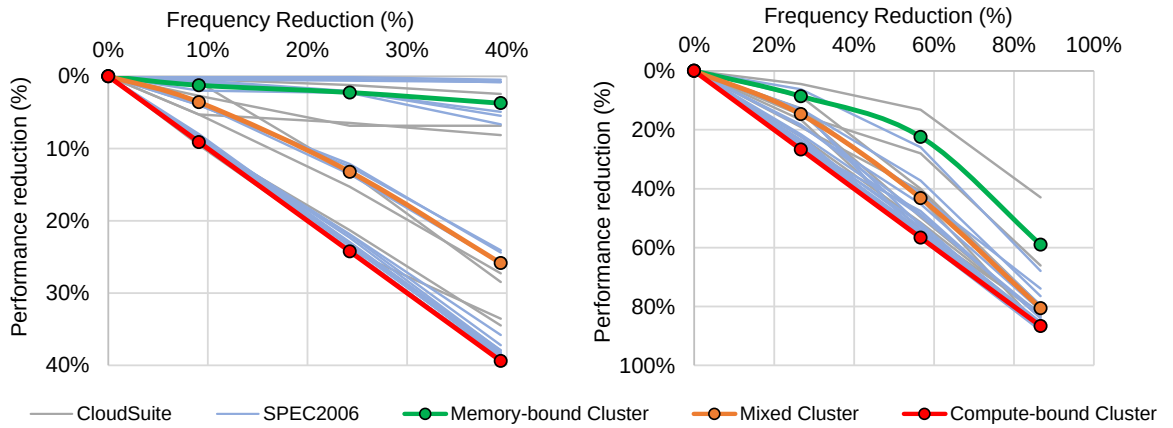


Figure 4.2: Classes of applications, in terms of performance sensitivity to frequency reduction for Skylake (*left*) and X-Gen 3 (*right*).

The performance of an application depends on the CPU operating frequency. However, different applications exhibit a different degree of sensitivity to frequency scaling. In our experiments, we measure the execution time of different applications for the four frequency points considered in the two platforms. We use 24 benchmarks from the SPEC CPU 2006 [20], and the whole CloudSuite benchmark suite [23] for these experiments. After extracting the performance profile for each application on each operating frequency point, we apply a K-Means clustering algorithm to form clusters of applications with common behavior.

In Figure 4.2, we observe the three main clusters of applications with distinctly different behavior that are identified by our experiments: a) *compute-bound*, b) *mixed*, and c) *memory-bound* applications. The first cluster (*compute-bound*) includes applications with performance degradation which scales almost linearly to CPU frequency reduction, with a ratio of approximately $1x$ for both CPU architectures. The second cluster (*mixed*) is less affected by the frequency reduction, however in the case of the ARM-based platform, an aggressive frequency reduction leads to a significant performance loss of up to 76%. The third cluster (*memory-bound*) includes applications that are relatively insensitive to frequency reduction for the Intel-based platform. For the X-Gen3 server, the performance degradation

is lower than the other two classes but higher than the Intel-based platform. An aggressive frequency reduction may again lead to a significant performance loss.

In our simulation experiments (described in Section 4.4) we assume three equiprobable performance classes for the application VMs, corresponding to the three clusters identified above. Depending on the class of VM_m , we apply the corresponding performance scaling behavior to transform the requested CPU capacity $CPU_{m,f_{max}}^{req}$ at the maximum CPU frequency f_{max} , to the equivalent CPU capacity $CPU_{m,f}^{req}$ at a lower frequency f . It should be noted that in a real setup the frequency scaling sensitivity of a job can be easily estimated at execution time, by using hardware performance counters to quantify the CPU intensiveness of the job.

4.4 Experimental Evaluation

In this section, we evaluate *XM-VFS* comparing it with a state-of-the-art policy [18]. Work in [18] exploits the VM consolidation techniques discussed in [19] combined with DVFS to optimize the allocation of the VMs on the nodes. We choose *LrMmt* (local regression minimum migration time) policy from [18] to compare against *XM-VFS*, because *LrMmt* outperforms other policies for VM consolidation in [18].

For our experiments, we perform simulations that are based on parameters experimentally quantified on real hardware and the results are, therefore, realistic. We experiment with two node architectures, with diverse characteristics: a) an Intel-based platform, which, as discussed in Section 4.3, has wide voltage margins enabling significant energy gains, and b) an ARM-based platform which has narrow voltage margins. We implemented our policies in the context of the CloudSim toolkit [25]. CloudSim is widely used, it already supports VM consolidation and it is configurable in terms of node hardware parameters. However, we had to extend CloudSim in the following ways:

- To support configuration at reduced voltage margins and to simulate node crashes when operating at reduced voltage margins configurations.
- To be able to parse and use Google traces [26] as input. CloudSim previously could only use PlanetLab traces [27] which are rather small-scale.
- To implement the cost model discussed in 4.1 and our new policy (*XM-VFS*).

As input to our simulations, we use a subset of Google traces consisting of 10,000 VMs. Each experiment simulates a full day. We use a scheduling period of 300 seconds. We also use real-world time series for VM pricing [17], ranging from \$0.0062 to \$0.896 per hour, depending on VM resource requirements agreed in the SLA. Further, we evaluate our policy using both a constant average energy price (\$0.14 per KWh) [57] and variant energy prices [85] where the energy price changes every 300 seconds.

Figure 4.3 shows the distribution of the energy price samples within 1 day in ascending sort. We consider as the low energy price the \$0.10 per KWh and as the high price the \$0.22 per KWh, which correspond to the 5th (yellow line) and the 95th (green line) percentile, respectively.

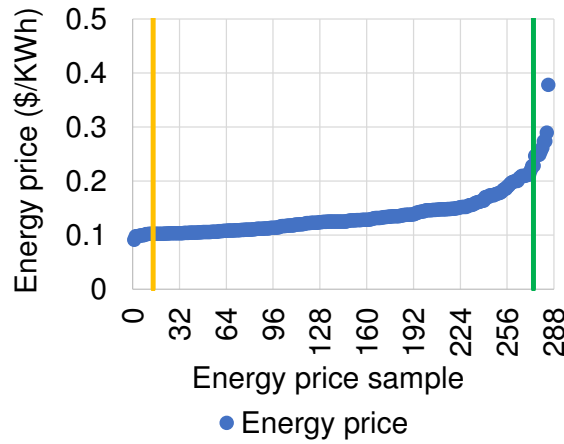


Figure 4.3: Distribution of energy prices during a single day.

In our experiments we assume a baseline failure rate for nodes operating at reduced margins, much higher than what we have observed during node characterization. Moreover, we assume the manifestation of all errors as complete node crashes. These are rather pessimistic assumptions, therefore we expect gains to be higher on real infrastructure.

4.4.1 Target node utilization sensitivity analysis

The target utilization is an important parameter that may affect the behavior of the scheduling policies and, consequently, the allocation of the VMs on the available nodes. On the one hand, a low target utilization results in suboptimally utilized hardware and limits opportunities to power-off nodes to increase the efficiency of the datacentre. On the other hand, a high target utilization may result in node overloads, triggering SLA violations and necessitating VM restarts to other nodes to mitigate overloads.

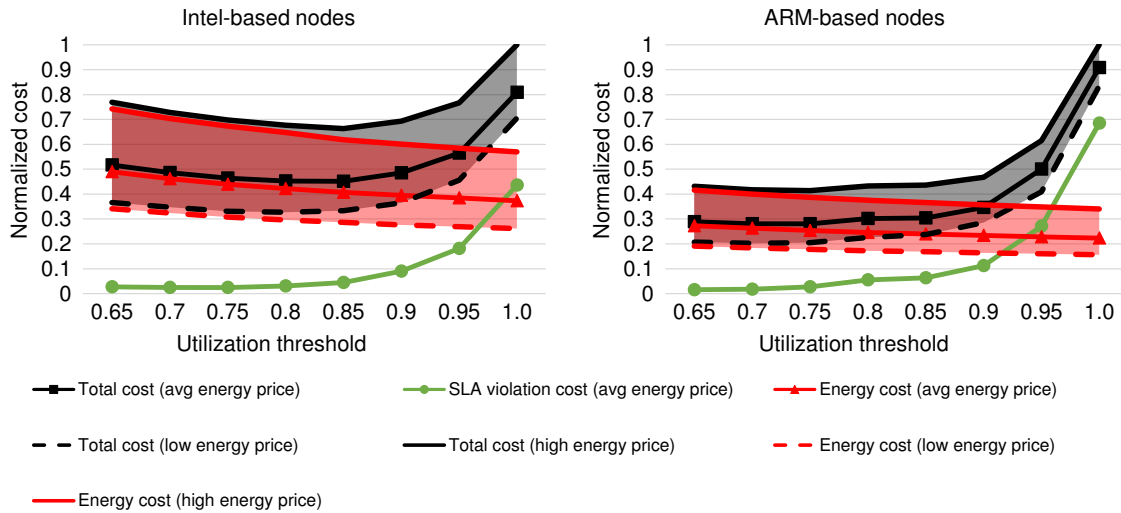


Figure 4.4: Target utilization threshold sensitivity analysis for the Intel (left) and ARM (right) -based nodes.

To quantify the effect of different target node utilization values on the effectiveness of our policy and *LrMmt*, we performed a sensitivity analysis for nominally configured nodes, both SLA violation penalty models and different energy prices. Figure 4.4 illustrates the results of this analysis for our policy on both platforms for the PPR SLA model. The lines illustrate the total cost, the energy cost and the SLA violation cost for different energy prices (low, average and high) for different node utilization targets. The shaded areas show the range of the cost between the low and the high energy price for the total cost, the energy cost and the SLA violation cost. All values are normalized w.r.t. the highest total cost (utilization threshold of 1.0 for both platforms) for the high energy price. Note that for the case of the SLA violation cost, the cost difference between low and high energy price is almost negligible.

We observe that for both platforms a higher utilization target may indeed lead to SLA violations due to overloads and VM restarts. On the other hand, a lower utilization threshold keeps the nodes underutilized, increasing, as expected, the energy cost. It is clearly advisable to avoid extreme utilization targets. However, there is a region around the optimal utilization threshold, where the total cost is not significantly affected. We observe that this region is more dependent on the platform, rather than the energy price. Similar observations hold for the *LrMmt* policy and the FPR SLA model as well. In our experiments, we use a utilization threshold target of 0.8 and 0.7 for the Intel and the ARM platform, respectively.

4.4.2 XM-VFS behavior at the baseline failure probability

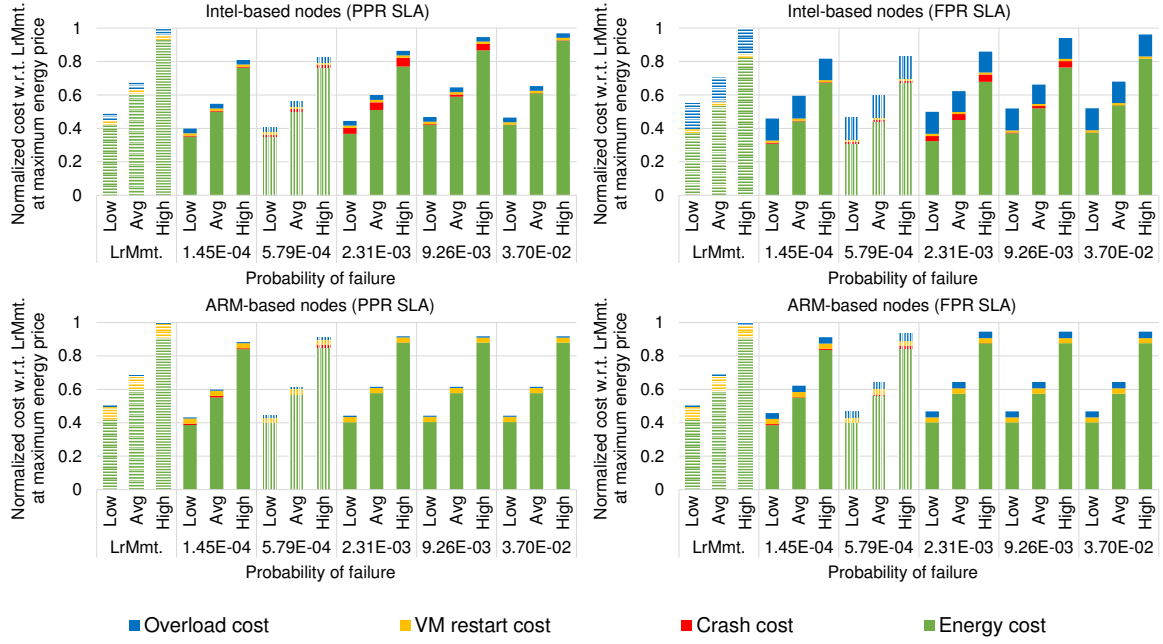


Figure 4.5: Parametric analysis of cost w.r.t. probability of failure with low, average and high energy prices for PPR and FPR SLAs on both platforms.

In this section we compare *XM-VFS* with *LrMmt* for both SLA models introduced in Section 4.1 considering a low, average and high energy price to illustrate the effectiveness of our policy.

Figure 4.5 illustrates the cost benefits of *XM-VFS* w.r.t. *LrMmt* for Intel and ARM-based nodes for both SLA models and different energy prices (low, average and high). We itemize costs as energy cost and SLA violation penalty cost due to VM restarts, node overloads, and node crashes. All costs are normalized w.r.t the total cost of the *LrMmt* policy for the high energy price (horizontally striped) for each case. The figures illustrate a parametric analysis of cost w.r.t. failure probability. The horizontal axis presents the probability of failure which increases between consecutive bars. The vertically striped bars correspond to the baseline failure probability we assumed.

The results indicate that *XM-VFS* successfully employs the cost function introduced in Section 4.1 to dynamically and adaptively react to the characteristics of the architecture and the workload and decide on datacentre configuration.

For the baseline failure probability, we observe that *XM-VFS* results in a lower total cost compared with *LrMmt* for the Intel-based nodes for both SLA models and for any energy

price. More specifically, the cost gain is very similar for each SLA penalty model at any energy price. Specifically, the cost is reduced by 17.24% and by 16.33% on average for the PPR SLA and FPR SLA models, respectively. As the Intel processor has wide voltage margins, our policy configures almost all the nodes at non-nominal settings, for any energy price, and reduces the energy consumption of the nodes by 17.72% on average for both SLA types. This benefit is not outweighed by the cost of potential SLA violation penalties due to crashes, which increase the total cost by 2.41% on average when operating at reduced margins. Therefore, *XM-VFS* is likely to configure nodes at reduced margins for any energy price for the Intel platform.

The overall cost reduction is less profound for the ARM-based platform. The X-Gene 3 processor has narrow voltage margins. Therefore, operation at reduced margins does not offer energy savings high enough to significantly outweigh penalties due to SLA violations for lower energy prices for the baseline failure probability. As a result, our policy decides to configure at reduced margins 17.01%, 39.51% and 75.22% of the nodes for the low, average and high energy price, respectively. The corresponding energy gains due to the configuration at reduced margins are 0.79%, 1.8% and 3.4%. However, the total cost gains are 10.77% and 7.2% on average for the PPR and FPR SLA model, respectively, which come mainly from the more frequent selection of the best operating point of each node n_i resulting to lower energy cost even in nominal conditions.

Even at the pessimistic baseline failure probability, the percentage of node crashes throughout all experiments did not exceed 0.37% for any scheduling period, while during most periods there were no crashes at all. Given this very low number of failures, the cost of node restarts is negligible (our model in Section 4.1 assumes this is zero). Also, the number of active nodes did not vary significantly across scheduling periods, avoiding needless powering-on and -off of the nodes. This is enabled by the increased elasticity of computational capacity configuration offered by combining reduced margins with built-in voltage-frequency (V, f) steps.

Finally, we observe that the most dominant parameters that affect the percentage of the nodes configured at reduced margins for a specific failure probability are the processor architecture and the energy price. Higher energy prices favor more aggressive configurations. The main difference between the SLA models in these experiments is that FPR SLA introduces higher penalties due to overloads, as the provider pays all the cost of the VM back to the cus-

tomers for the duration of a node overload. Moreover, regardless of the SLA model, our policy manages to reduce the cost of VM restarts, by triggering fewer VM restarts, which occur at the expense of more overloads. Overall, *XM-VFS* manages to reduce the cumulative cost due to node overloads and VM restarts compared with *LrMmt*. There is obviously a trade-off between VM restarts and node overloads. A policy may decide to tolerate node overloads in order to reduce VM restarts, or opt for more VM restarts to reduce node overloads. *XM-VFS* has more opportunities to reduce VM restarts, by operating at a higher performance (V, f) step whenever necessary, while at the same time limiting the cost increase associated with the higher (V, f) step by exploiting operation at reduced margins.

4.4.3 XM-VFS behavior at different failure probabilities

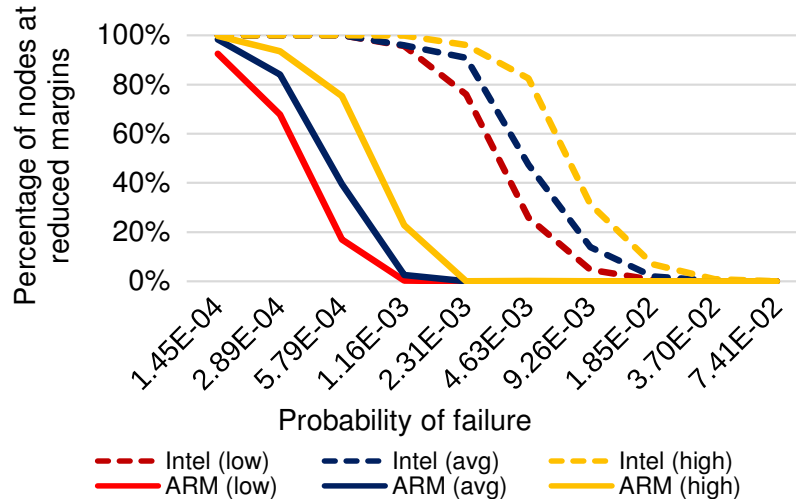


Figure 4.6: Percentage of nodes at reduced margins for both platforms.

In the previous discussion, we assumed a baseline pessimistic failure probability, when nodes operate at non-nominal configurations exploiting voltage margins. In this section, we perform a sensitivity analysis on the behavior of *XM-VFS* on both architectures with different failure probabilities, both higher and lower than the baseline as illustrated in Figure 4.5. Moreover, Figure 4.6 shows the percentage of the nodes configured at reduced margins for different failure probabilities for the PPR SLA model. Each line corresponds to a different combination of the energy price and platform and shows the percentage of the nodes that were configured at reduced margins for each failure probability.

The first important observation is that SLA violations due to crashes tend to decrease, as we move to lower or higher failure probabilities (Figures 4.5). Low failure probabilities

expectedly result in fewer node crashes. Higher failure rates, on the other hand, make *XM-VFS* configure fewer nodes (and beyond a point practically 0 nodes, as illustrated by the lines in Figure 4.6) at non-nominal settings, thus again decreasing crashes and the respective SLA violation penalties. On the ARM platform, the improvement of energy efficiency by operating at reduced margins is limited, therefore the policy generally configures fewer nodes in non-nominal settings at any given failure probability compared with the Intel-based datacentre. The total cost also increases with the increase of the failure probability. The increase is gradual for Intel nodes. For the ARM nodes, it reaches its peak and stabilizes sooner (at a lower failure probability), as *XM-VFS* opts sooner to operate all nodes at nominal settings.

Moreover, Figure 4.6 shows that the energy price affects the percentage of nodes configured at reduced margins (excluding the failure probabilities where this percentage converges to 0% or 100%, as the failure probability is the dominant parameter at these cases). As it is expected, *XM-VFS* configures more nodes at reduced margins for a higher energy price, as the energy gains outweigh the cost due to potential crashes. On the other hand, a lower energy price results in fewer nodes being set at reduced margins for a specific failure probability, as there is less room for energy gains due to the lower energy price. In any case, *XM-VFS* adapts in any energy price and ubiquitously outperforms *LrMmt* for both platforms and at all failure probabilities and energy prices.

4.4.4 **XM-VFS behavior with varying energy pricing**

In this section, we study the behavior of *XM-VFS* with varying energy pricing. In contrast with the previous sections where the energy price was constant for each experiment, in these experiments, the energy price changes at the beginning of each scheduling period.

Moreover, Figure 4.8 presents the normalized cumulative workload of the VMs for each period and the energy price per period. Figure 4.7 illustrates the percentage of the nodes at reduced margins for different failure probabilities with variant energy prices. Each area corresponds to a specific failure probability, and it shows the percentage of the nodes at reduced margins for each scheduling period, assuming this failure probability.

The first observation is the different behavior for the two platforms through the periods, even for the same failure probabilities. The Intel platform has wide voltage margins, therefore our policy opts to configure at reduced margins more nodes than the ARM platform for the same failure probability (except from the highest, unrealistic probabilities, for which the

percentage of nodes at reduced margins converges to 0% for both platforms). We also observe that for a wide range of failure probabilities (excluding the lowest and highest failure probability) the percentage of the nodes at reduced margins depends on the energy price. For the lowest and highest failure probabilities considered in our study, this percentage converges to 100% or 0%, respectively.

Namely, for the Intel-based nodes, we observe that for the baseline and lower failure probabilities almost all the nodes are configured at reduced margins for any energy price (note that the baseline failure probability in Figure 4.7 overlays the lower ones, as the percentage of the nodes at reduced margins for these probabilities converges to 100%). For the ARM-based nodes, this happens only for the highest energy prices and only for the lowest failure probability. Moving to higher failure probabilities, there is a range that is sensitive to the energy price (0.0023 - 0.0185 and 0.000289 - 0.0016 for the Intel and ARM platforms, respectively). We observe that *XM-VFS* adapts to each energy price changing the percentage of the nodes configured at reduced margins according to the energy price. For probabilities beyond these ranges, our policy configures the nodes at nominal settings mostly, irrespectively of the energy price.

One more interesting observation comes from the periods 40 - 120, where the energy price is almost constant, however the percentage of the nodes at reduced margins fluctuates through periods for both platforms. This happens because the algorithm considers the workload, beyond the energy price. During these periods the workload fluctuation is high, resulting in different cost-optimal node configurations at each period.

In conclusion, we observe that our policy manages to adapt to changes related to the energy price or the workload pressure, increasing or decreasing dynamically the percentage of the nodes at reduced margins when this is needed, considering the trade-off between energy cost and potential SLA violation costs.

4.5 Related Work

All previous research efforts try to reduce datacentre energy consumption while operating the nodes at nominal conditions. However, undervolting [16, 21] the processor in a diligent way may offer more energy savings to the plug. As we discussed earlier, in a scale-out environment these energy savings come at the educated risk of increasing the failure probability,

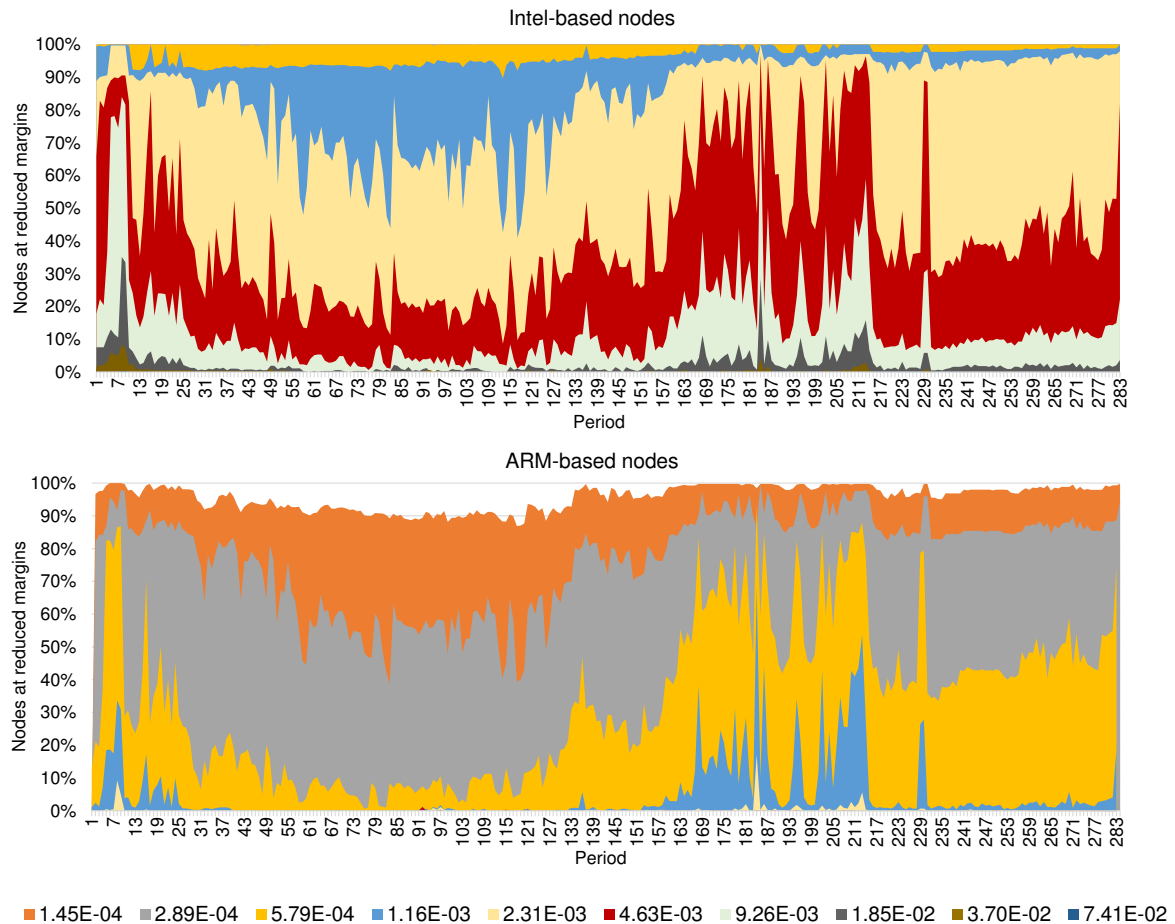


Figure 4.7: Percentage of the nodes at reduced margins with variant energy prices for different failure probabilities for the Intel and ARM platforms.

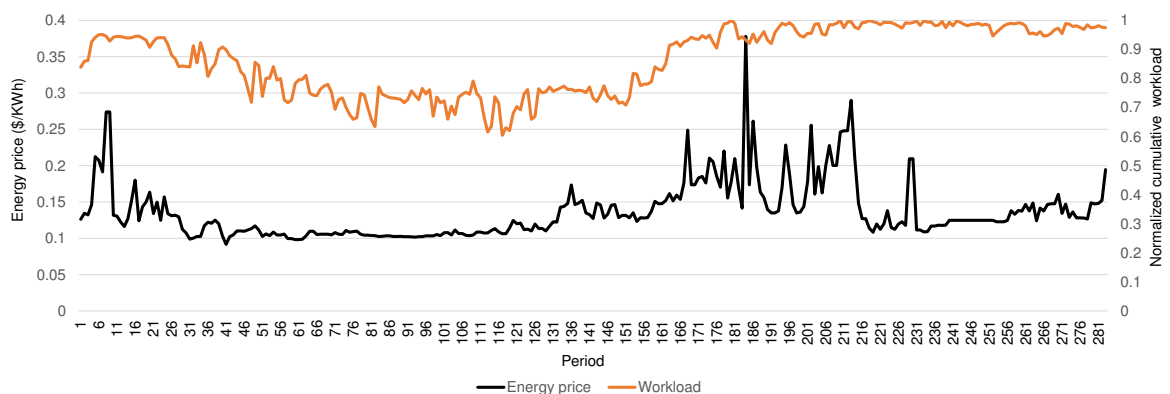


Figure 4.8: Normalized cumulative workload of the VMs and energy prices per period.

which may in turn lead to increased SLA violation penalties due to node crashes. No previous work studies the trade-off between energy savings and SLA violation penalties if the processor is configured at reduced margins.

VM consolidation is a common technique used to improve energy efficiency. VMs are

mapped to as few nodes as possible, in order to provide opportunities for unused nodes to be put to sleep state. However, consolidation may affect the quality of service (QoS) VMs enjoy to the extent of breaking contractual agreements (SLAs) and incurring penalties for the infrastructure provider. The problem of VM consolidation is further complicated by the dynamically changing workloads and resource requirements. The authors in [19,29,40] introduce heuristics to deal with this complexity when performing VM consolidation. The work in [98] introduces a scheduler with fast and high-quality decisions to identify the best suited resources for a job to increase performance. More recent works [28,54] introduce optimized VM consolidation algorithms which also target the reduction of VM migrations. [56] is one of the few policies which, similarly to ours, takes into account that jobs which run in a datacentre have different characteristics (CPU- or IO-bound tasks).

The consideration of energy price variations can reduce the cost. Work in [86] introduces an approach of geographical distribution of the jobs exploiting the variations of the energy price at different locations to reduce the cost, considering also the contribution of the cooling to the cost. The authors in [88] and [87] also consider geographical load balancing due to different energy prices, and they study the reduction of the cost due to use of renewable energy instead of brown energy. The works in [89] and [90] examine also the use of renewable energy considering also energy price variations, but in a single datacentre, aiming to increase the use of green energy respecting the task deadlines. Finally, the authors in [91] and [92] study variations in energy price for the wholesale market for different locations, introducing policies that reduce the cost while preserving the quality of service.

Modern processors support DVFS, which enables operation of the processor at different nominal (V, f) operating points and different points of performance / power consumption. [41] introduces a VM allocation algorithm for a DVFS-enabled cluster. The authors in [42] exploit DVFS to reduce energy consumption for the execution of tasks with deadline constraints.

A number of recent research efforts combine VM consolidation with DVFS. This combination has the potential to provide significant energy gains, without significantly penalizing the QoS. Works in [18,53] evaluate the benefits of applying VM consolidation in synergy with DVFS. They extend the policies in [19] in order to integrate DVFS. The authors in [18] report average energy savings up to 37.86% compared with previous work in [19]. As discussed in Section 4.4, our policy outperforms [18].

To the best of our knowledge our work is the first that minimizes cloud infrastructure operator costs by carefully operating the processor outside its nominal operating envelope, considering the educated risk of node failures and the associated SLA violation penalties. On top of operation at reduced margins, we also employ VM consolidation and DVFS. Moreover, we consider that VMs with different characteristics (CPU-bound or IO-bound) behave differently when executed at different clock frequencies. We also take into account the variations of energy pricing during the day (in our experiments in periods of 5 minutes), allowing the policy to adapt its placement and configuration decisions to minimize the cost. Moreover, we use two different SLA models to evaluate our work. The first one is more mild as it only considers the amount of CPU resources that were not allocated for a VM. The second one is stricter as the cloud provider has to pay a penalty based on the full price of the VM for the duration of the violation. Experimentation with different SLA violation penalty models shows the flexibility of the policy. As discussed in Section 4.4, our policy manages to adapt to different SLA models and to the variations of the energy price, and finally, it outperforms a state-of-the-art policy [18] for both SLA models and different energy prices.

Chapter 5

Adaptive utilization threshold for Cloud Computing

In this chapter we focus on the question "Can a policy that calculates the target utilization of the nodes at runtime outperform policies that assume *a priori* knowledge of the workload using offline characterization to derive an optimal static target utilization?". More specifically, we calculate the utilization threshold of each node independently for each period analyzing the workload characteristics observed in the past and predicting short-term future behavior. For this calculation, we quantify the correlation among colocated workloads on each node in order to identify a sweet-spot between high node utilization and risk of SLA violation penalties due to unexpected load increases, which may lead to node overloads. The main contributions of our work are the following:

- We introduce a scalable, analytic and architecture-agnostic model to estimate target CPU utilization for individual nodes in a datacentre, exploiting knowledge on the past behaviour of the VMs scheduled on these nodes. These adaptive utilization targets favor high VM consolidation, while controlling the risk of node overloads.
- We find that, in order to achieve this dual goal, the policy needs to identify correlations between the CPU capacity requests of co-scheduled VMs and consider them when deciding on node utilization targets.
- The proposed method works fully online, without *a priori* knowledge regarding the workload of the datacentre, nor requiring any pre-training.

- We introduce an adaptive VM management and node configuration policy that exploits the aforementioned adaptive thresholds for VM consolidation, together with VFS and CPU configuration at reduced voltage margins. The policy also considers the cost of SLA violation penalties due to both node overloads and crashes due to operation at reduced margins.
- We perform simulations to evaluate our approach using real datacentre workload traces and realistic node parameters ((V, f) steps, voltage margins, power consumption to the plug, failure probability) derived from real systems with two different processor architectures, an Intel Xeon E3-1120 v5, and an ARM-based Ampere Computing X-Gen 3.

To the best of our knowledge, this is the first work that combines per node target CPU utilization adaptivity with VFS and relaxed CPU voltage guardbands in a cost-centric approach. The optimization goal is to increase the profit of cloud infrastructure providers exploring the trade-off between energy cost and potential SLA violation penalties. Our results show that our adaptive policy manages to always outperform two state-of-the-art policies using statically optimal target CPU utilization values derived via pre-characterization of the workload at hand on the respective architecture: (a) a policy that combines VM consolidation with VFS [18], and (b) our policy in Chapter 4, which also exploits reduced CPU voltage guardbands on top of VM consolidation and VFS.

5.1 Background

In this section, we briefly outline the notation and some basic assumptions used in our adaptive model to be introduced in Section 5.2. We use a notation and assumptions compatible to those in Chapter 4. Also, Table 5.1 summarizes the variables we use in Section 5.1 and Section 5.2.

Table 5.1: Summary of variables.

Variable	Description
A	A platform-specific constant for the static power consumption.
B	A platform-specific constant for the dynamic power consumption.

$\hat{CPU}_{i,m,f_i,p+1,k}^{alloc}$	The estimated allocated CPU capacity of VM_m at f_i for the k^{th} timeslot of $p + 1$.
$CPU_{m,f_i,p}^{req}$	The requested CPU capacity of VM_m at f_i for p .
$\hat{CPU}_{m,f,p+1}^{req}$	The estimated CPU requirements of VM_m at f for $p + 1$.
$\overline{CPU}_{m,f,p}^{req}$	The mean CPU requirements of VM_m at f over the past p periods.
$CPU_{m,f_{max}}^{req}$	The requested CPU capacity of VM_m at f_{max} .
CPU_{m,f_i}^{SLA}	The requested CPU capacity of VM_m defined in SLA at f_i .
$CPU_{m,f_{max}}^{SLA}$	The requested CPU capacity of VM_m defined in SLA at f_{max} .
$\hat{cost}E_{i,V_i,f_i,p+1}$	The estimated energy cost related to n_i at (V_i, f_i) for $p + 1$.
$\hat{cost}E_{i,V_i,f_i,p+1}^{evct}$	The estimated energy cost of all the evicted VMs from n_i for $p + 1$.
$\hat{cost}E_{i,V_i,f_i,p+1}^{rem}$	The estimated energy cost of n_i at (V_i, f_i) for $p + 1$ for the VMs that remain at the node.
$\hat{cost}E_{i,m,f_i,V,f,p+1}^{vm}$	The estimated energy cost of an evicted V_m from n_i if the node operated at (V, f) for $p + 1$.
$\hat{cost}V_{m,f_i,p+1,k}^{vm}$	The expected SLA violation cost VM_m for the k^{th} timeslot of $p + 1$.
$\hat{cost}V_{i,f_i,p+1}$	The total estimated SLA violation cost of n_i at f_i for $p + 1$.
$\hat{cost}_{i,V_i,f_i,p+1}$	The expected cost of n_i at (V_i, f_i) for $p + 1$.
f_i	The frequency of the processor.
f_{max}	The frequency of the processor at the highest performance nominal operating point.
k	The k^{th} timeslot.
K	The number of timeslots of a scheduling period.
$\hat{load}_{i,f_i,p+1}$	The estimated load related to n_i at f_i for $p + 1$.
$\hat{load}_{i,f_i,p+1}^{evct}$	The estimated load of the evicted VMs from n_i (which operates at f_i) for $p + 1$.
$\hat{load}_{i,f_i,p+1,f}^{vms}$	The estimated load of the evicted VMs from n_i (which operates at f_i) for $p + 1$ for each supported frequency f .
$\hat{load}_{i,f_i,p+1}^{rem}$	The estimated load of n_i at f_i for $p + 1$ based on the VMs that will remain on the node.
M	The number of the VMs.
MEM_i	The memory amount of n_i .

MEM_m^{SLA}	The requested memory of VM_m defined in SLA.
N	The number of nodes.
n_i	The i^{th} node.
$nodes_{f,p}$	The number of nodes operated at f during p .
$nodes_p$	The number of active nodes during p .
P_{fail}	The probability of failure at reduced voltage margins configuration.
p	The scheduling period.
$prdT$	The total duration of a scheduling period.
$price_{en}$	The energy price per time unit.
$price_m$	The VM_m price per time unit.
q_m	A multiplier that increases the severity of SLA penalty for VM_m .
R_m	The number of timeslots required for VM_m for rescheduling and restart.
$slotT$	The duration of each timeslot.
V_{max}	The supply voltage of the processor at the highest performance nominal operating point.
V_i	The supply voltage of the processor.
VM_m	The m^{th} VM.
$\hat{u}_{i,f_i,p+1}$	The estimated CPU utilization of n_i at f_i for $p + 1$.
$u_{i,f,p}^{target}$	The target CPU utilization of n_i for f at p .

5.1.1 System model

We assume a datacentre with nodes $n_i, 1 \leq i \leq N$. Each node has limited memory MEM_i . All nodes have the same CPU, which can operate at different nominal voltage-frequency points (V, f) , where (V_{max}, f_{max}) offers maximum performance. For each nominal point, there is a reduced voltage configuration (V^r, f) with $V^r < V$, offering the same performance at lower power consumption. However, in this case the node may fail with probability $P_{fail} > 0$, whereas $P_{fail} = 0$ for nominal configurations. We assume crash failures, where the node stops working and all hosted VMs stop running. Modeling and tolerating silent data corruptions that might occur due to CPU undervolting is beyond the scope of this work.

Different jobs are submitted for execution, packaged as virtual machines $VM_m, 1 \leq m \leq M$ with formally declared resource requirements in respective SLAs. Let MEM_m^{SLA} be the

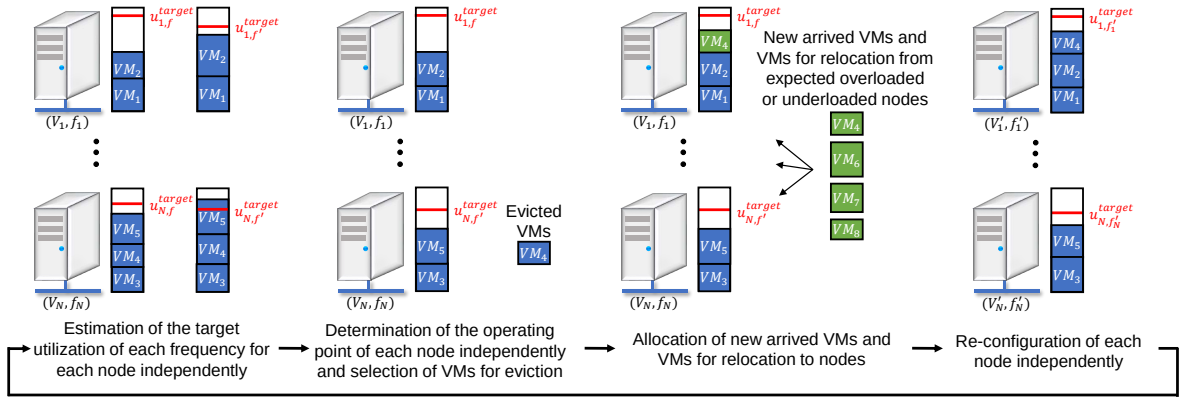


Figure 5.1: Overview of the proposed approach.

required memory and $C_{m,f_{max}}^{SLA}$ be the CPU capacity at f_{max} for VM_m as per its SLA. Notably, when the CPU operates at $f < f_{max}$, the effective CPU capacity as per the SLA becomes $C_{m,f}^{SLA} \geq C_{m,f_{max}}^{SLA}$ (the increase depends on the sensitivity of the VM to frequency scaling).

We partition time into periods and assume that VMs arrive and terminate at their boundaries. Further, we divide each period in K timeslots of duration $slotT$ (the total duration of a period is $prdT = K \times slotT$). Let $C_{m,f,p}^{req}$ be the actual CPU capacity requirements of VM_m for frequency f throughout period p . Note that $C_{m,f,p}^{req} \leq C_{m,f}^{SLA}$, i.e., in some periods the VM may not need the full CPU capacity declared in its SLA.

Let $C_{m,f,p,k}^{alloc}$ be the CPU capacity at frequency f that is allocated to VM_m in the k^{th} timeslot of p . If the host node is overloaded, the VM may get less capacity than required, $C_{m,f,p,k}^{alloc} < C_{m,f,p}^{req}$. Furthermore, the restart of VM_m on a node in period p (due to an eviction from its old host) takes R_m time slots (linearly dependent on the contractually agreed memory footprint of VM_m) thus $C_{m,f,p,k}^{alloc} = 0, 1 \leq k \leq R_m$. We assume VMs to be stateless, therefore they can be restarted without needing elaborate migration. Also, if the host of VM_m crashes (due to operation with reduced voltage), $C_{m,f,p,k}^{alloc} = 0, 1 \leq k \leq K$ for all slots of the period.

5.1.2 VM allocation approach

When a period ends, the target utilization, VM allocation and operating point is decided for each node, so as to minimize the expected cost for the datacentre due to the node's energy consumption and SLA violation penalties – the latter apply in case of node overload as well as in case the VM remains unavailable because it is being restarted on another node or the host node crashes.

Our approach is divided in the following main steps, illustrated in Figure 5.1. (1) We esti-

mate the target node utilization for each CPU frequency based on the current VM allocation and load, in order to identify expected node overloads for each operating point and pick the VMs that need to be evicted from the node to avoid overload. (2) We select the best nominal operating point for each node considering the trade-off between the node's energy cost and SLA violation penalties due to restart of the evicted VMs on other nodes. (3) We allocate evicted VMs and newly arrived VMs to nodes, respecting the selected operating point and target utilization of each node. At this point, we consider activating additional nodes or deactivating underutilized nodes. (4) We re-configure each node considering the updated VM allocation, exploiting operation at reduced voltage margins for additional cost gains. The operating point is selected in a cost-driven manner. Note that steps (1), (2) and (4) are performed using only local data of the node. Thus, the approach is scalable to large datacentres.

5.2 Problem Modeling

We introduce a model that considers the trade-off between the energy cost and potential SLA violation penalties. For the completeness of the model, we explain each part of it, although a few parts has similarities to the model of section 4.1. In contrast with the approach in Chapter 4, this model does not require in advance knowledge and characterization of the workload, but instead quantifies and adapts to workload characteristics at execution time. Moreover, it estimates the cost at the level of the entire system, even though scheduling decisions are taken at the node level. Hence, our approach is: (a) easily applicable in realistic settings, (b) scalable, and (c) cost-effective even for large datacentres. In this section we discuss in detail the estimation of the different costs involved, while in Section 5.3 we focus on the adaptation to workload characteristics.

5.2.1 Target CPU utilization and VM evictions

Each node n_i has a target CPU utilization $u_{i,f,p}^{target}$ for n_i during period p for CPU frequency f . This serves as a conservative upper bound for the CPU load to be assigned to the node, so as to avoid node overload in case some of the hosted VMs request more CPU capacity than expected. When the period ends, the target utilization is adjusted for the next period $p + 1$ based on the current VM allocation and a tentative frequency f_i , possibly also selecting some VMs for eviction to avoid overloads.

The tentative VM evictions are captured via matrix $Evct$, where $Evct[i, f_i, m, p+1] = 1$ if VM_m is to be evicted from n_i assuming operation at f_i in $p+1$, else 0. Similarly, we use $Rem[i, f_i, m, p+1]$ to encode whether VM_m will remain on n_i in $p+1$. This exploration is done for each of the supported CPU frequencies, in order to find the best option for n_i . Note that any VM allocation must respect the full memory requirements of each hosted VM, i.e., $\sum_{m=1}^M Rem[i, *, m, p+1] \times MEM_m^{SLA} \leq MEM_i$.

5.2.2 Load estimation

We estimate the CPU requirements of each VM_m for frequency f in the next period $p+1$ as $\hat{C}_{m,f,p+1}^{req} = \bar{C}_{m,f,p}^{req}$, where $\bar{C}_{m,f,p}^{req}$ are the VM's mean CPU requirements over the past periods up to p adjusted to f .

Then, the estimated load for n_i assuming operation at f_i , based on the VMs that will remain on the node during the next period, is calculated as

$$\hat{load}_{i,f_i,p+1}^{rem} = \sum_{m=1}^M Rem[i, f_i, m, p+1] \times \hat{C}_{m,f_i,p+1}^{req} \quad (5.1)$$

We also estimate the load of each VM to be evicted from n_i and restarted on another node. However, we do not know the hosts of these VMs nor their respective CPU frequencies for the next period. As an approximation, we estimate the load of the evicted VMs for each supported frequency f as

$$\hat{load}_{i,f_i,p+1,f}^{vms} = \sum_{m=1}^M Evct[i, f_i, m, p+1] \times \hat{C}_{m,f,p+1}^{req} \quad (5.2)$$

Then, we estimate the total load of the VMs to be evicted from n_i for the entire system, by weighing the above load for each frequency according to the nodes' operating frequencies during the previous period p

$$\hat{load}_{i,f_i,p+1}^{evct} = \sum_{f \in F} \frac{nodes_{f,p}}{nodes_p} \times \hat{load}_{i,f_i,p+1,f}^{vms} \quad (5.3)$$

where $nodes_{f,p}$ is the number of nodes that operated at f during p , and $nodes_p$ is the number of nodes that were active during p .

Thus, the total estimated VM load associated with node n_i , assuming the node will operate at f_i during the next period $p+1$, is

$$\hat{load}_{i,f_i,p+1} = \hat{load}_{i,f_i,p+1}^{rem} + \hat{load}_{i,f_i,p+1}^{evct} \quad (5.4)$$

5.2.3 Energy cost estimation

The estimated energy cost of each active node n_i that will host one or more VMs during the next period is calculated as

$$\hat{cost}E_{i,V_i,f_i,p+1}^{rem} = P_{node}(V_i, f_i, \hat{u}_{i,f_i,p+1}) \times prdT \times price_{en} \quad (5.5)$$

$$P_{node}(V, f, u) = A + B \times f \times V^2 \times u \quad (5.6)$$

$$\hat{u}_{i,f_i,p+1} = \min(\hat{load}_{i,f_i,p+1}^{rem}, 1) \quad (5.7)$$

assuming it will be configured to operate at (V_i, f_i) , where $price_{en}$ is the energy price and $\hat{u}_{i,f_i,p+1}$ is the estimated node utilization for the next period based on the estimated load as discussed in Section 5.2.2. Function $P_{node}(V, f, u)$ returns the node's power consumption when the CPU operates at (V, f) with utilization u , based on the linear power estimation model discussed in Section 4.3.3 where A and B are platform-specific constants for the node's static and dynamic power consumption, respectively.

The energy cost of VMs that will be evicted from n_i is approximated as follows. First, the individual cost of each evicted VM_m if hosted on a node configured to operate at (V, f) is

$$\hat{cost}E_{i,m,f_i,V,f,p+1}^{vm} = P_{vm}(V, f, \hat{C}_{m,f,p+1}^{req}) \times prdT \times price_{en} \quad (5.8)$$

$$P_{vm}(V, f, u) = \frac{u}{\bar{u}_{f_i,p}^{target}} \times A + B \times f \times V^2 \times u \quad (5.9)$$

In this case, the CPU utilization factor for the dynamic component of the power estimation function (second term of Equation 5.9) is set to the expected CPU load of VM_m during the next period $\hat{C}_{m,f,p+1}^{req}$. We also factor the static power consumption component (first term of Equation 5.9) with the ratio of that load to the mean target CPU utilization of all nodes configured to operate at f_i during the previous period p ($\bar{u}_{f_i,p}^{target}$). This ratio represents the probability that a particular VM eviction leads to the activation of a new node in $p + 1$.

Then, we estimate the total energy cost of all VMs that will be evicted from n_i , as

$$\begin{aligned} \hat{cost}E_{i,V_i,f_i,p+1}^{evct} &= \sum_{(V,f) \in VF} \left(\frac{nodes_{V,f,p}}{nodes_p} \right) \\ &\times \sum_{m=1}^M Evct[i, f_i, m, p + 1] \times \hat{cost}E_{i,m,f_i,V,f,p+1}^{vm} \end{aligned} \quad (5.10)$$

weighing the contribution of each operating point in the same spirit this done for load estimation in Equation 5.3.

Thus, the total estimated energy cost related to node n_i assuming it will operate at (V_i, f_i) in the next period $p + 1$ is

$$\hat{cost}E_{i,V_i,f_i,p+1} = \hat{cost}E_{i,V_i,f_i,p+1}^{rem} + \hat{cost}E_{i,V_i,f_i,p+1}^{evct} \quad (5.11)$$

We assume that if a node fails, it does so for an entire period, during which it does not run any VMs and does not consume any energy.

5.2.4 SLA violation cost estimation

We estimate the CPU capacity $\hat{C}_{i,m,f_i,p+1,k}^{alloc}$ allocated to VM_m hosted on node n_i during the k^{th} timeslot of period $p+1$ as follows. If $n_i \text{ load}_{i,f_i,p+1}^{rem} \leq 1$ then $\hat{C}_{i,m,f_i,p+1,k}^{alloc} = \hat{C}_{i,m,f_i,p+1}^{req}$, else (if the node is overloaded) $\hat{C}_{i,m,f_i,p+1,k}^{alloc} < \hat{C}_{i,m,f_i,p+1}^{req}$ since the node's CPU capacity is divided among the VMs proportionally to their requirements (all VMs are assumed to have the same priority). Also, $\hat{C}_{i,m,f_i,p+1,k}^{alloc} = 0$ during timeslots when VM_m is restarting as well as during all timeslots of the period if n_i crashes.

Whenever VM_m does not get the requested CPU capacity, the datacentre provider has to pay a penalty to the customer. We consider the *Percentage Price Refund (PPR)* model [11] where the penalty is a percentage of the VM price for the duration of the violation, depending on the percentage of the CPU capacity that was not allocated w.r.t. to the SLA requirements. More specifically, the expected violation cost for VM_m for the k^{th} timeslot of period $p + 1$ is

$$\hat{cost}V_{m,f_i,p+1,k}^{vm} = \begin{cases} q_m \times \frac{C_{m,f_i}^{SLA} - \hat{C}_{m,f_i,p+1,k}^{alloc}}{C_{m,f_i}^{SLA}} \times price_m, & \text{if } \hat{C}_{m,f_i,p+1,k}^{alloc} < \hat{C}_{m,f_i,p+1}^{req} \\ 0, & \text{if } \hat{C}_{m,f_i,p+1,k}^{alloc} = \hat{C}_{m,f_i,p+1}^{req} \end{cases} \quad (5.12)$$

where q_m represents a constant penalty factor for each VM, which is defined in the SLA and controls (increases or decreases) the cost of the penalty, and $price_m$ is the price charged for executing VM_m per timeslot.

We then calculate the total estimated SLA violation cost for n_i for the entire period $p + 1$, as follows

$$\hat{cost}V_{i,f_i,p+1} = \sum_{m=1}^M \sum_{k=1}^K Rem[i, f_i, m, p + 1] \times \hat{cost}V_{m,f_i,p+1,k}^{vm} \quad (5.13)$$

Equation 5.13 captures the cost of all the violation penalties. More specifically, in Equation 5.12 we define $\hat{cost}V_{m,f_i,p+1,k}^{vm}$ as $\hat{cost}V_{i,f_i,p+1}^{nofail}$, if there are no node crashes, and the penalty cost comes from node overloads and VM restarts. In the case of a node crash, we define it as $\hat{cost}V_{i,f_i,p+1}^{fail}$.

5.2.5 Total estimated node cost

Based on the above, the total estimated operational cost related to n_i , assuming it will operate at (V_i, f_i) during the next period $p + 1$ is

$$\begin{aligned} \hat{cost}_{i,V_i,f_i,p+1} = & P_{fail} \times \hat{cost}V_{i,f_i,p+1}^{fail} + \\ & (1 - P_{fail}) \times (\hat{cost}E_{i,V_i,f_i,p+1} + \hat{cost}V_{i,f_i,p+1}^{nofail}) \end{aligned} \quad (5.14)$$

calculated by weighing the cost of normal node operation vs. a node crash with the corresponding probabilities.

5.3 Adaptive target CPU utilization

5.3.1 System model

The target node utilization for the next period $u_{i,f,p+1}^{target}$ is set with some slack, so that the node can handle load beyond the estimated $\hat{C}_{i,f_i,p+1}^{req}$ without being overloaded. This is subject to an interesting trade-off: High utilization targets result in more energy-efficient hence also greener and lower cost operation. However, if future CPU requests of the VMs are underestimated and overshoot the CPU capacity of the node, SLA violations will occur, incurring penalties for the operator.

In addition, potential correlation between different collocated VMs further perplexes the estimation of a good CPU utilization target. Let $C_{m,p}^{ratio} = \frac{C_{m,f_{max},p}^{req}}{C_{m,f_{max}}^{SLA}}$ be the normalized CPU requirements of VM_m during period p with respect to the contractually promised CPU capacity, both at frequency f_{max} . Figure 5.2 illustrates the C^{ratio} of eight VMs from Google traces [26] over 10 periods. Each line corresponds to a different VM. One can identify clusters of VMs with similar, positively correlated $C_{m,p}^{ratio}$ variations (lines of the same color). Moreover, VMs in the blue and green cluster are characterized by a rapid increase of their requested CPU capacity at period 10. Finally, the red cluster follows the opposite trend of the blue one.

If decisions on target CPU utilization do not consider such phenomena, they are highly likely to lead to node overloads. On the one hand, the identification of positively correlated VMs should encourage more conservative values of target CPU utilization. On the other hand, the co-execution of anti-correlated VM clusters on the same node can protect against

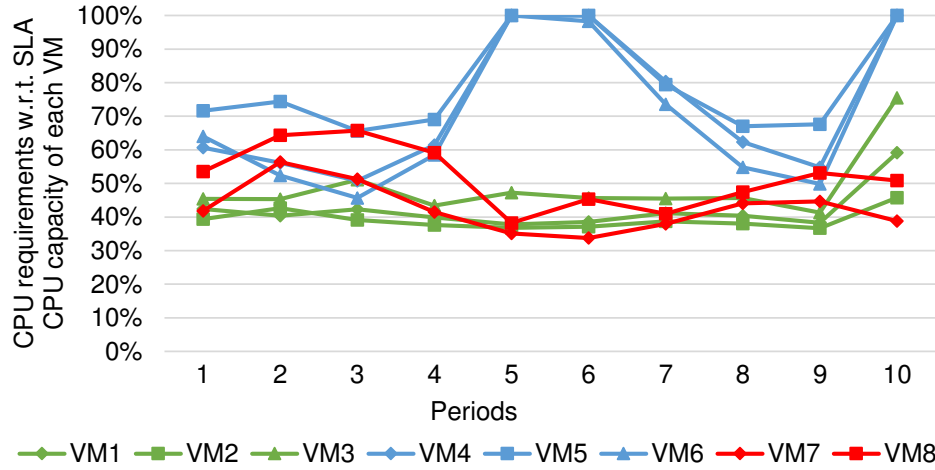


Figure 5.2: CPU pressure of different VMs w.r.t their contractual (SLA) maximum CPU requirements, over consecutive periods.

overloads due to dynamic workload changes: when one cluster increases its requests, its counterpart tends to decrease them, thus balancing the total load on the host node. This allows for more aggressive (higher) CPU utilization targets, and a more energy-efficient usage of the available resources.

To address the aforementioned challenges, at period boundaries we evaluate, separately for each node n_i , the past behavior of its assigned VMs and decide a new value for $u_{i,f,p+1}^{target}$ for the following period. The estimation process involves the following steps: (1) For every pair (VM_l, VM_m) of VMs on the node we quantify the Pearson correlation $corr_{(l,m),p}$ between the C^{ratio} of VM_l and VM_m during periods up to p . (2) We use linear regression to estimate the CPU capacity requests of every VM_m on the node, given the capacity requests of a different VM_l for the next period, according to $corr_{(l,m),p}$. (3) We pessimistically assume that on the next period a specific VM_l will request its maximum contractually agreed CPU capacity, i.e., $\hat{C}_{l,p+1}^{ratio} = 1$. Given this assumption and the correlations determined in step (2), we reach a conservative estimation of the respective total node load. (4) We repeat steps (2)-(3) using each VM on the node as the baseline for the estimation, and then use the highest (most pessimistic) load estimation for period $p + 1$ to set the new target CPU utilization for the node. Below we provide more details about these steps.

In step (2) we first apply linear regression, as modelled by Equation 5.15, to predict the C^{ratio} of VM_m in period $p + 1$ assuming we know the C^{ratio} of VM_l on the same period:

$$\hat{C}_{(m|l),p+1}^{ratio} = \alpha_{(m|l),p} \times \hat{C}_{l,p+1}^{ratio} + \beta_{(m|l),p} + \epsilon_{(m|l),p} \quad (5.15)$$

where $\alpha_{(m|l),p}$ is the slope of the regression line, $\beta_{(m|l),p}$ is the intercept, and $\epsilon_{(m|l),p}$ is the error of the prediction. We calculate $\alpha_{(m|l),p} = \text{corr}_{(m,l),p} \times \frac{\text{std}_{m,p}}{\text{std}_{l,p}}$, where $\text{std}_{l,p}$ and $\text{std}_{m,p}$ are the standard deviations of the C^{ratio} of VM_l and VM_m respectively, over past periods up to p . To derive $\beta_{(m|l),p}$, we first calculate the mean C^{ratio} of VM_m and VM_l up to period p , let $\bar{C}_{m,p}^{ratio}$ and $\bar{C}_{l,p}^{ratio}$, respectively. Then, we set $\epsilon_{(m|l),p} = 0$ and solve Equation 5.15 for $\beta_{(m|l),p}$, using $\bar{C}_{m,p}^{ratio}$ and $\bar{C}_{l,p}^{ratio}$ in place of $\hat{C}_{(m|l),p+1}^{ratio}$ and $\hat{C}_{l,p+1}^{ratio}$. Finally, since our goal is to avoid node overloads, we conservatively set $\epsilon_{(m|l),p}$ to the maximum absolute error between the predicted values of our model and the observed real values over past periods.

In step 3, we predict the load of n_i for period $p+1$, as a fraction of its total CPU capacity, assuming $\hat{C}_{l,p+1}^{ratio} = 1$, as

$$\hat{load}_{i,l,f_i,p+1}^{pre} = C_{l,f_i}^{SLA} + \sum_{m \neq l} \text{Rem}[i, f_i, m, p+1] \times \hat{C}_{(m|l),f_i,p+1}^{pre} \quad (5.16)$$

where $\hat{C}_{(m|l),f_i,p+1}^{pre}$ is the predicted CPU capacity request by VM_m for frequency f_i under the assumption that $\hat{C}_{l,p+1}^{ratio} = 1$. This is calculated by deriving the predicted CPU capacity request $\hat{C}_{(m|l),f_{max},p+1}^{pre} = \hat{C}_{(m|l),f_{max},p+1}^{ratio} \times C_{m,f_{max}}^{SLA}$ at f_{max} and then calculating $\hat{C}_{(m|l),f_i,p+1}^{pre}$ for f_i (according to the sensitivity of VM_m to frequency scaling).

Finally, in step 4, after performing the above for each VM hosted on n_i , we determine the highest (most pessimistic) load across all scenarios $\hat{load}_{i,f_i,p+1}^{max}$. We then use this value to set the new target CPU utilization for n_i at f_i in $p+1$:

$$u_{i,f_i,p+1}^{target} = u_{i,f_i,p}^{target} \times \left(1 + \frac{1 - \hat{load}_{i,f_i,p+1}^{max}}{\hat{load}_{i,f_i,p+1}^{max}}\right) = \frac{u_{i,f_i,p}^{target}}{\hat{load}_{i,f_i,p+1}^{max}} \quad (5.17)$$

where we adjust the previous target based on the relative slack $\frac{1 - \hat{load}_{i,f_i,p+1}^{max}}{\hat{load}_{i,f_i,p+1}^{max}}$ between the most pessimistic load and full node utilization, asymptotically pushing the maximum expected CPU requests on the node close to (yet lower than) full utilization. A negative slack indicates that the node is at risk of an overload, thus the target utilization is lowered. If, on the other hand, the slack is positive, there is CPU capacity available for hosting more VMs, and the target utilization is increased. As an exception, if $u_{i,f_i,p+1}^{target} > \hat{load}_{i,f_i,p+1}^{max}$, we set $u_{i,f_i,p+1}^{target} = u_{i,f_i,p}^{target}$. This guards against the corner-case of having enough spare CPU capacity on n_i just because it did not host "enough" VMs in p , as in this case a further increase of the target utilization would introduce the risk of excessive arrival of additional VMs on n_i , leading to an overload.

5.4 VM Allocation and Node Configuration Policy

In this section, we introduce Adaptive Target Utilization and Configuration (*ATUC*), a VM allocation policy which dynamically adjusts the target utilization and configuration of each node to increase the profit of datacentre providers. The optimization target is to minimize the predicted cost-to-load ratio of each node for the next period

$$\hat{C}L_{i,V_i,f_i,p+1}^{ratio} = \frac{\hat{cost}_{i,V_i,f_i,p+1}}{\hat{load}_{i,f_i,p+1}} \quad (5.18)$$

where the nominator and denominator of the fraction comes from Equation 5.14 and Equation 5.4, respectively. Note that this involves only local information, as discussed in Sections 5.2.2, 5.2.3, and 5.2.4.

The *ATUC* policy consists of the following main steps:

1) Estimation of target CPU utilization for each frequency: We use the methodology discussed in Section 5.3 to estimate the new target utilization for each node independently. More specifically, we estimate $u_{i,f_i,p+1}^{target}$ for the next period $p + 1$, for each node n_i and for each supported CPU frequency step f_i . This step is executed independently on each node with complexity $\mathcal{O}(1)$.

2) Cost-driven node configuration and VM eviction: Given $u_{i,f_i,p+1}^{target}$ for each f_i , we select the best operating point for each node n_i independently in $p + 1$ and the VMs to be evicted in this case. At this point, we focus on VM (re)assignment and only consider nominal operating points ($Pfail = 0$). The decision whether to configure the node to operate at reduced voltage margins (where $Pfail > 0$) is taken in the last step of the policy.

The high-level logic of this step is explained in Algorithm 3. The algorithm takes as input the VMs hosted on n_i during the previous period p , and the node utilization targets for each CPU frequency during the next period $p + 1$. The output is the nominal operating point and the VMs to be evicted from n_i for that period. The logic is briefly as follows. For each nominal CPU operating point, starting from the previous VM allocation (lines 3-4) and the corresponding estimated load (line 5), the VMs with the smallest SLA violation cost due to restart on another node (lines 7-14) are tentatively selected for eviction (lines 15-16) and the expected node load is recalculated (line 17), until the load does not exceed the target utilization. Note that the evicted VMs are assumed to be hosted on a non-overloaded node (the cost only comes from the time slots needed to restart the VM). Then, the operating point in question is preferred if the respective cost-to-load ratio is better than the one of the

Algorithm 3 Selection of nominal operating point (V'_i, f'_i) and evicted VMs for n_i in the next period $p + 1$.

Input: $vmsHost_{i,p}, u_{i,*}^{target}$

Output: $(V'_i, f'_i), vmsEvct_{i,p+1}$

```

1:  $minCLRatio \leftarrow \infty$ 
2: for each nominal  $(V_i, f_i)$  and  $u_{i,f_i,p+1}^{target}$  do
3:    $Rem[i, f_i, m, p + 1] \leftarrow 1, \forall m \in vmsHost_{i,p}$ 
4:    $Evct[i, f_i, m, p + 1] \leftarrow 0, \forall m \in vmsHost_{i,p}$ 
5:    $\hat{load}_{i,f_i,p+1}^{rem} \leftarrow$  calculate based on Equation 5.1
6:   while  $\hat{load}_{i,f_i,p+1}^{rem} > u_{i,f_i,p+1}^{target}$  do
7:      $minRestartCost \leftarrow \infty$ 
8:     for each  $m : Rem[i, f_i, m, p + 1] = 1$  do
9:        $vmRestartCost \leftarrow \sum_{k=1}^{R_m} costV_{m,f_i,p+1,k}^{vm}$ 
10:      if  $vmRestartCost < minRestartCost$  then
11:         $minRestartCost \leftarrow vmRestartCost$ 
12:         $vmEvct \leftarrow m$ 
13:      end if
14:    end for
15:     $Rem[i, f_i, vmEvct, p + 1] \leftarrow 0$ 
16:     $Evct[i, f_i, vmEvct, p + 1] \leftarrow 1$ 
17:     $\hat{load}_{i,f_i,p+1}^{rem} \leftarrow$  calculate based on Equation 5.1
18:  end while
19:   $\hat{CL}_{i,V_i,f_i,p+1}^{ratio} \leftarrow$  calculate based on Equation 5.18
20:  if  $\hat{CL}_{i,V_i,f_i,p+1}^{ratio} < minCLRatio$  then
21:     $minCLRatio \leftarrow \hat{CL}_{i,V_i,f_i,p+1}^{ratio}$ 
22:     $(V'_i, f'_i) \leftarrow (V_i, f_i)$ 
23:  end if
24: end for
25:  $vmsEvct_{i,p+1} \leftarrow \{m : Evct[i, f'_i, m, p + 1] = 1\}$ 

```

current selection (19-22). The final (best) selection is the result of performing this check for all nominal operating points. The complexity is $\mathcal{O}(1)$, as the number of the VMs on the nodes does not depend on the total number of the VMs in the datacentre, and each node has a certain capacity for accommodating VMs and this number is relative small.

3) Assignment of unallocated VMs to nodes and switch-off of underutilized nodes:

At this point there are two types of unallocated VMs that need to be assigned to nodes: the VMs selected for eviction in the previous step and newly arrived VMs (if any). In a first step, we use the approach in Section 4.2 (best fit decreasing allocation) to assign unallocated VMs to nodes with $\mathcal{O}(m \log m)$ complexity w.r.t. the number of the VMs that will be allocated on the nodes. In a second step, we adopt the technique in [19] to identify severely underutilized nodes and switch them off, by relocating the VMs accommodated on them (as above), and the complexity is $\mathcal{O}(n \log n)$ (n is the number of the nodes in the datacentre).

4) Cost-driven re-configuration of nodes at a nominal or reduced voltage margins operating points: Finally, based on the new VM allocation, the CPU of each node is configured to the most cost-effective operating point. This is done by examining all (V, f) configurations that do not violate the target utilization, and selecting the one that minimizes the respective cost-to-load ratio for each node independently according to Equation 5.18. In this step, reduced voltage configurations are considered as well, considering the probability of node crash and the respective SLA violation costs as per Equation 5.14. This step is executed independently on each node, and the complexity is $\mathcal{O}(1)$.

5.5 Experimental Evaluation

In this section, we discuss the evaluation of the *ATUC* policy through simulations. For these experiments, we use real-world traces and hardware parameters derived from real platforms. We also compare *ATUC* against *LrMmt* [18], which combines VM consolidation with VFS, and *XM-VFS* (Chapter 4), which exploits CPU configuration at reduced voltage margins on top of these two techniques. Both these policies employ a static target CPU utilization value, which is derived by offline characterization of the workload on the platform architecture at hand.

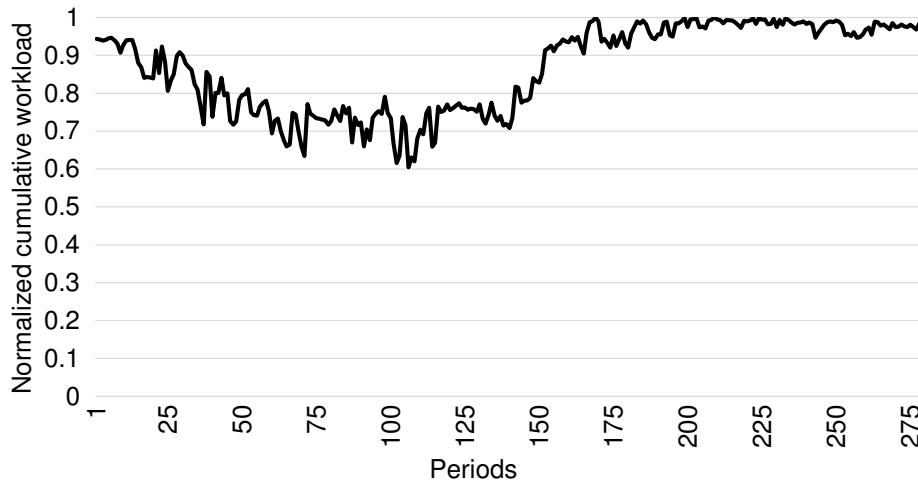


Figure 5.3: Cumulative CPU load (normalized w.r.t the maximum cumulative CPU load) variation in time (across different periods).

5.5.1 Experimental Setup

For our simulations, we use the CloudSim toolkit [25], a popular framework for evaluating cloud policies, which already supports mechanisms such as VM consolidation and configurable node hardware parameters. We extend CloudSim to implement our system model (Section 5.2) and the *ATUC* policy (Section 5.4). Those extensions include target utilization adaptivity at the granularity of individual nodes (Section 5.3).

We use real values for the energy price [57] (\$0.14 per KWh) and the range price of the VMs [17] (\$0.0062 – \$0.896 per hour). Our simulated workload consists of 10,000 VMs from Google traces [26], for a period of one day. Figure 5.3 illustrates the cumulative CPU load for this workload for each period. The values are normalized to the maximum cumulative CPU load during the simulated period. This dataset combines different interesting characteristics. During the first half of the day, it is characterized by a low frequency CPU load variation, combined with significant high frequency fluctuations. On the second half, the cumulative load does not vary. However, the CPU requests of individual jobs still follow different, varying patterns — not visible in the figure. The mean cumulative normalized CPU load is 0.86, with a standard deviation of 0.12.

We assume the policy is invoked every 300 seconds. We opt for the same scheduling period as in [18], which is also the default for CloudSim. The execution time of the policy is negligible, particularly considering that job scheduling in large scale cloud systems is performed at orders of magnitude lower frequency compared with the OS scheduling frequency

Table 5.2: Intel Xeon E3-1220 v5 and ARM-based Ampere Computing X-Gene 3 operating points.

Intel platform			ARM platform		
Frequency (GHz)	Nominal Voltage (mV)	Reduced Voltage (mV)	Frequency (GHz)	Nominal Voltage (mV)	Reduced Voltage (mV)
2.0	850	666	0.4	880	790
2.5	922	741	1.3	880	790
3.0	1075	865	2.2	880	830
3.3	1147	929	3.0	880	840

on individual nodes.

5.5.2 Hardware parameters

We configure the simulator with the hardware parameters of two real systems with different processor architectures, namely an Intel Xeon E3-1220 v5 and an ARM-based Ampere Computing X-Gene 3. In order to enable a direct and fair comparison between *ATUC* and *XM-VFS*, we use the parameters identified by the characterization of these two systems in Section 4.3, in terms of (a) the reduced voltage margins of the processors, (b) failure probability when operating at reduced margins configurations, (c) a power model that estimates the power at the plug for different utilization levels and operating points, and (d) the performance sensitivity of applications due to frequency scaling.

More specifically, Table 5.2 summarizes the parameters of the (V, f) pairs (nominal and reduced margins) used for the Intel and the ARM platforms. The failure probability for a scheduling period of 300 seconds, when the processors operate at reduced voltage margins configurations, is $P_{fail} = 0.000579$. This value was experimentally derived in Section 4.3 after running the applications for 6 consecutive days without observing any node crashes. We again assume crash failures (no silent data corruptions). When the CPU is at nominal conditions, it operates with a negligible failure probability. Power at the plug, is calculated using linear model $A + B * V^2 * f * u$, where A and B are platform-specific constants of the model, V is the voltage, f is the frequency and u is the utilization of the processor. For

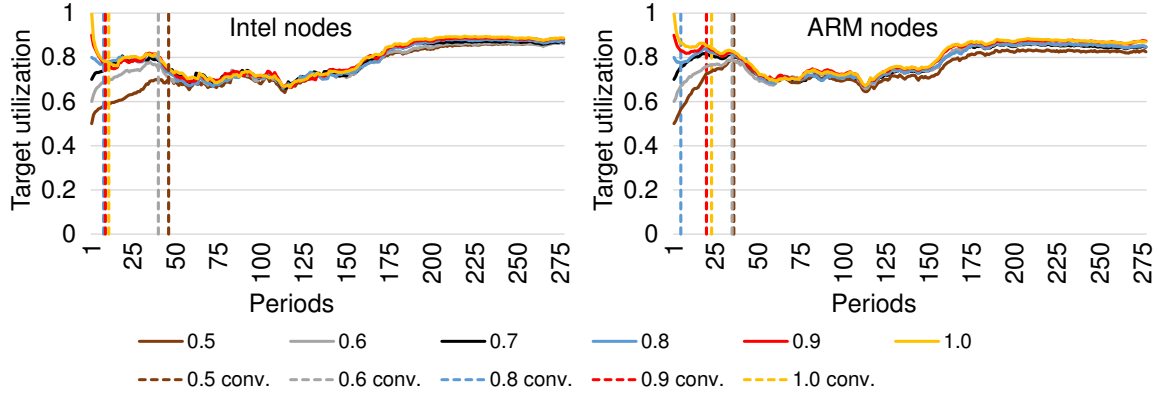


Figure 5.4: $\overline{u^{target}}$ per period for different initial values of u^{target} (0.5 – 1.0).

the Intel platform $A = 34.01$ and $B = 18.98$, whereas for the ARM platform $A = 52.48$ and $B = 34.38$ (Section 4.3).

5.5.3 Threshold and cost convergence

ATUC adjusts the target utilization of each node independently, adapts VM allocation to nodes, and hardware configuration according to the dynamic workload requirements, in order to minimize the cost for the operator. It is important for *ATUC* to be resilient to sub-optimal initial values of the target utilization (u^{target}), as historical data of VM behavior are not available during the first scheduling periods. Therefore, at this first set of experiments, we select several initial values for the target utilization, and we study whether and with what delay this initial target utilization converges to more cost-efficient target utilization values. We select 0.7 as the baseline for the convergence study, as it is approximately in the middle of the range of evaluated initial target utilization values.

Figure 5.4 presents $\overline{u^{target}}$ (the mean target utilization of the nodes) for each period for different initial u^{target} values (different solid lines). The dashed lines mark (for the initial u^{target} with the same color) the time of convergence with the $\overline{u^{target}}$ timeseries for an initial target utilization of 0.7.

We observe that *ATUC* is resilient to the “sub-optimal” choice of the initial value of u^{target} , as it eventually converges to similar $\overline{u^{target}}$ values in all cases. However, the convergence latency depends on that initial value. Lower initial values (0.5 and 0.6) converge later, as *ATUC* is conservative when increasing the target utilization, in order to avoid node overloads. For example, recovering from a low initial u^{target} of 0.5 requires 46 and 36 periods for the Intel and ARM platform, respectively. On the other hand, the convergence from higher

initial values of u^{target} (0.8, 0.9 and 1.0) is quick, as $ATUC$ decreases the target utilization rapidly when at risk for node overloads. As an example, $\overline{u^{target}}$ estimations reached with initial $u^{target} = 1.0$ and 0.7 (baseline) converge within just 11 and 23 periods for the Intel and ARM nodes, respectively. Overall, $ATUC$ adjusts the target utilization within the ranges of 0.64 - 0.89 and 0.64 - 0.88 for the Intel and the ARM, respectively.

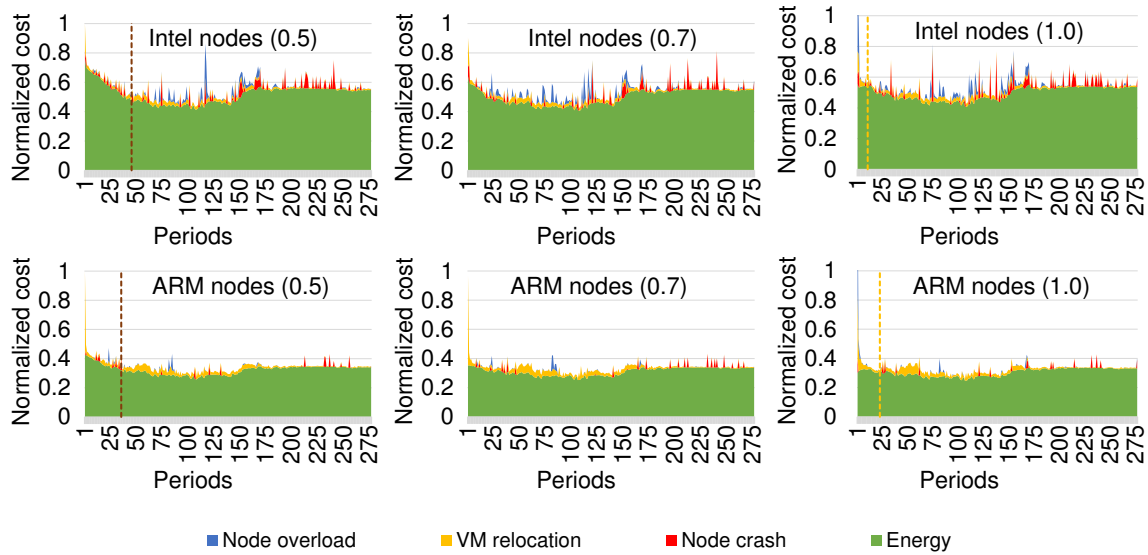


Figure 5.5: Normalized cost per period for initial values of target utilization of 0.5, 0.7 and 1.0.

Apart from the resilience of $\overline{u^{target}}$ convergence to the initial estimate, we also study the cost breakdown per period for different initial u^{target} values, to evaluate the cost of convergence. We perform experiments for low (0.5), high (1.0) and the baseline (0.7) initial values for u^{target} . Figure 5.5 illustrates the results, where the green, yellow, blue and red areas represent the cost of energy, and the cost of SLA violations due to VM relocation, node overloads and node crashes, respectively. All costs have been normalized, for each of the two architectures (Intel and ARM), w.r.t. the total cost during the first period of execution with an initial $u^{target} = 0.5$. Similarly to the previous figure, for the experiments with initial $u^{target} = 0.5$ and 1.0 we use vertical dashed lines to illustrate the convergence time of the target utilization timeseries w.r.t to that for 0.7.

As expected, the case of an initial u^{target} of 0.5 introduces a higher cost (dominated by the energy cost) for both platforms, compared with the other two cases (0.7 and 1.0) for the periods before convergence of the target utilization, as the resulting low utilization target leads to the activation of more nodes. On the other hand, although a high u^{target} (1.0) achieves the

lowest energy cost during the first periods, the total cost is higher due to node overloading. However, once convergence is reached, we observe that the energy cost pattern is similar, irrespective of the initial u^{target} selection. More specifically, after convergence, the relative energy cost difference between algorithm invocations with initial estimates 0.5 and 1.0, and the invocation with 0.7 is in the order of 1.76% and 1.55% respectively for the Intel platform. The respective relative energy cost differences for the ARM platform are 2.3% and 1.65%. In conclusion, *ATUC* manages to recover from sub-optimal initial values of the target utilization, asymptotically introducing similar energy cost per period, irrespective of the value of target utilization used for the initialization of the adaptive estimation algorithm. In the next section, we focus on the total cost, also including SLA violation penalties.

5.5.4 *ATUC* total cost of operation

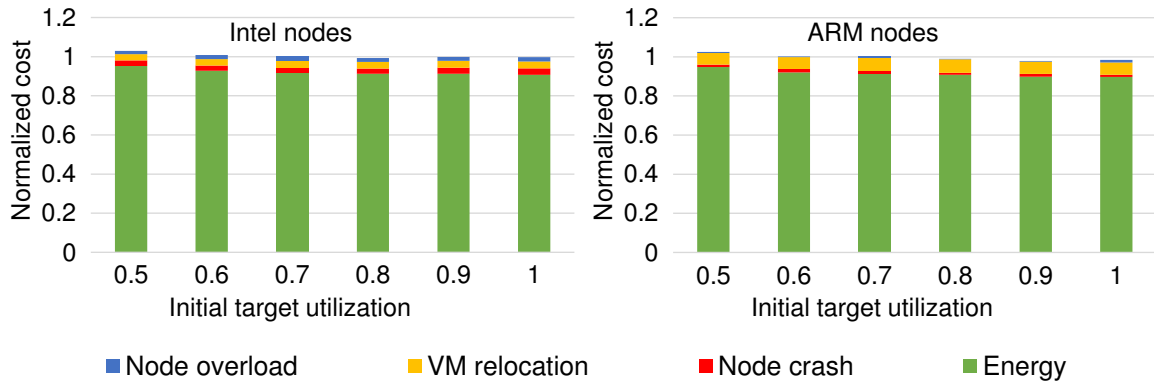


Figure 5.6: Normalized (total) cost for different initial values of u^{target} .

As we discussed in Section 5.5.3, *ATUC* manages to recover from sub-optimal initial values of u^{target} achieving, after convergence, a low relative energy cost difference w.r.t. the baseline initial u^{target} case. In this section, we evaluate the effectiveness of *ATUC* regarding the total cost, also including the periods of sub-optimal operation until convergence. Figure 5.6 illustrates the results. Similarly to Section 5.5.3, we itemize the cost as energy cost and SLA violation penalty cost (node crash, VM relocation, node overload). For each architecture, cost is normalized w.r.t. the cost achieved by setting the initial value of $u^{target} = 0.7$.

We observe that a low initial value (0.5) results in the highest cost, for both platforms. This is expected, as in this case *ATUC* (i) requires the longest time to recover from this sub-optimal initial target utilization value, and (ii) during this time the estimated target CPU utilization of nodes remains lower than needed to achieve overload-safe operation for the architecture and

workload at hand, hence the policy activates a higher number of nodes, which consume more energy. However, even in this case, the total cost is only 2.98% and 2.46% higher compared with the baseline initial value for the Intel and ARM nodes, respectively. For all other initial values, we observe that the cost is similar, for both platforms. These results confirm that *ATUC* manages to recover not only quickly, but also cost-effectively from adverse values for the initial estimate of u^{target} .

5.5.5 Comparison of *ATUC* with state-of-the-art policies

In this section, we compare *ATUC* with *XM-VFS* and *LrMmt* in terms of energy and cost efficiency. The statically optimal target CPU utilization threshold is a function of the workload and the performance elasticity of the architecture along different (V, f) operating points. Our approach in Chapter 4 has experimentally confirmed this; after brute-force evaluation of different target utilization thresholds, it was determined that both *XM-VFS* and *LrMmt* introduce the minimum cost at the essentially the same target CPU utilization threshold, which is equal to 0.8 on Intel nodes and 0.7 on ARM nodes. In this work, we focus on evaluating whether *ATUC* can match the efficiency of these policies, which need in advance characterization of the workload to derive the optimal static target utilization. Such a characterization is unrealistic for several real-world settings. The performance of these policies is highly dependent on the quantification of an appropriate static target utilization for the specific workload at hand. If the workload is not characterized in advance, or if it is characterized on a non-representative sample, performance is significantly penalized, for deviations of the target node utilization from the statically optimal value. For example for the workload in our evaluation, *XM-VFS* may introduce 14.23% higher cost on Intel nodes for a static target utilization of 0.65 (instead of the optimal 0.8).

Thanks to the effective adaptation employed by *ATUC*, the initial u^{target} value does not significantly affect cost (Section 5.5.4), therefore we set it at 0.7 for all these experiments. Figure 5.7 presents the results of the comparison between *ATUC* and the two state-of-the-art policies for the Intel and ARM platforms. Once again, we itemize the cost incurred due to energy consumption and the different sources of SLA violation penalties. All costs are normalized w.r.t. the total cost of the policy we compare against on the corresponding platform.

We observe that *ATUC* is not inferior to policies requiring in advance knowledge of workload characteristics. Instead, it manages to outperform them in all cases, even compared with

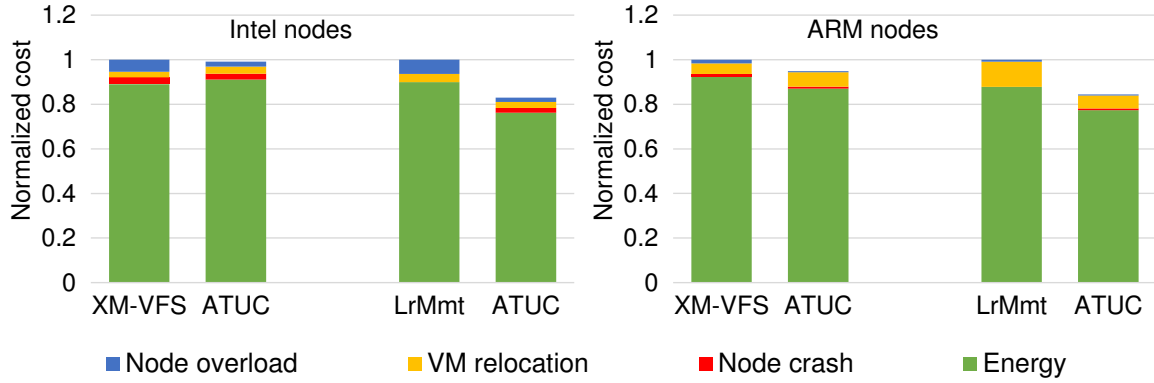


Figure 5.7: Cost comparison of *ATUC* with *LrMmt* and *XM-VFS*.

XM-VFS which also exploits reduced CPU voltage margins.

Specifically, with *ATUC* the percentage of penalty cost due to node overloads over the total cost is 2.27% for Intel and 0.51% for ARM nodes on average. The respective contribution of overload penalty to the total cost is higher for both the other policies: with *XM-VFS* it accounts for 5.41% (Intel) and 1.67% (ARM) of the total cost, whereas with *LrMmt* for 6.39% (Intel) and 0.85% (ARM).

ATUC reduces the SLA violation penalties due to overloads at the expense of a slightly higher energy cost compared against *XM-VFS* for the Intel platform, however, still achieving 0.85% cost gains against a policy that uses an optimal static target utilization for the current workload. In the case of the ARM platform, beyond reducing SLA penalties due to overloads, *ATUC* also achieves energy gains over *XM-VFS*, resulting in 5.1% net cost gains. Moreover, *ATUC* manages to reduce the cost of crashes (red area) by 19.65% and 29.52% compared with *XM-VFS* for the Intel and ARM platform, respectively.

Comparing *ATUC* with *LrMmt*, we observe that *ATUC*, beyond reducing SLA violation costs due to node overloading, also reduces the penalties due to VM relocations managing to adapt to the dynamic workload requirements. There are also energy gains against *LrMmt*, however, given that — unlike *ATUC* and *XM-VFS* — *LrMmt* does not exploit configuration at reduced margins, these energy gains come mainly thanks to operation at reduced margins. The cost gains of *ATUC* against *LrMmt* are 17.06% for the Intel and 15.61% for the ARM nodes.

In summary *ATUC* manages, with zero-knowledge of the workload, to dynamically identify favorable points in the energy reduction vs. SLA penalties trade-off. As a result, it outperforms *XM-VFS* and *LrMmt*, which use a statistically optimal target utilization, obtained

via offline pre-characterization of the workload. The effective, adaptive estimation of the target CPU utilization is a key contributor towards the gains achieved by *ATUC*. In turn, a key feature of the estimation process is quantifying and considering the correlation between colocated workloads. If this correlation is overlooked, the operating cost achieved by *ATUC* is 24.52% higher due to the larger number of node overloads.

5.6 Related Work

Making educated decisions on the trade-off between energy cost and potential SLA violation penalties is a challenging, yet necessary undertaking towards cost reduction for cloud infrastructure providers. In Chapter 4.5, we discussed that VM consolidation [19, 29, 40] is a commonly used technique that tries to consolidate VMs to fewer nodes (without triggering SLA violations) to create opportunities to switch nodes off.

Handling the aforementioned trade-off is further complicated by the dynamic variations of typical cloud workloads, which – if not guarded-against through overallocation of resources – may affect the quality of service (QoS) received by the end-users, increasing the SLA violation cost. The work in [55] uses Reinforcement Learning (RL) to adapt to a dynamic environment, while reducing the energy consumption and preserving the performance of the system. Authors in [58] discuss an approach to satisfy the maximum response time requirements in multi-tier applications for Cloud, detecting and resolving bottlenecks and predicting the optimal configuration. The study in [59] uses fuzzy logic to estimate a lower and upper utilization threshold of the nodes, which is used to characterize nodes as underloaded or overloaded, and to optimize the VM placement in order to reduce energy consumption and SLA violation penalties. The work in [99] introduces a scheduler that exploits information of previously seen applications to discover similarities with existing applications and schedule them to servers with minimum interference with other applications. Authors in [60] deal with the dynamic workload problem, proposing two regression-based algorithms to estimate the upper utilization threshold of the nodes and detect overloads, combined with an algorithm that selects VMs from overloaded nodes, minimizing the VM migration time. In contrast with [19] and [60], our policy – apart from detecting node overloads – considers the target utilization of the nodes for the mapping of VMs to nodes, both for new VMs and for re-located ones. Moreover, unlike policies in [19] and [60], *ATUC* does not employ arbitrary

safety parameters, which directly affect the energy vs. SLA penalty costs trade-off.

Also, in Chapter 4.5, we discussed that apart from VM consolidation, works that combine VM management with DVFS [41] can exploit opportunities for additional energy gains, as DVFS allows processors to operate at more power-efficient states. The study in [61] introduces an algorithm that employs an adaptive threshold estimation to improve the performance through task replication, while saving energy through DVS. As in Chapter 4.4, we compare our approach with [18], where the authors combine VM consolidation with DVFS, to achieve significant energy gains compared with [19]. Also, we compare with our approach in Chapter 4 that combines VM consolidation and DVFS with CPU configuration at reduced voltage margins, aggressively reducing energy at the expense of an increased probability of crashes. However, similarly to *LrMmt*, the best performing policy in [18], these policies exploit a static CPU utilization target estimated through offline workload characterization. Our work combines adaptivity, VM consolidation, DVFS and CPU operation at reduced margins in a cost-driven profit maximization approach. It also exploits online analysis of past VM behavior, rather than offline pre-characterization, to reduce the risk of costly, hard to predict and mitigate node overloads.

Chapter 6

DRAM operation at reduced margins for profit increase of cloud providers.

In this chapter we focus on the question "Can operation at reduced margins of the DRAMs operating parameters (voltage and RP) increase the profit of Cloud infrastructures providers?". We reduce the margins of operating parameters such as the voltage and the RP of the DRAMs in order to investigate the effect on the energy consumption of the nodes. More specifically, we combine this technique with VM consolidation to examine the trade-off between the energy cost and potential SLA violation penalties due to the increased probability of increased DRAM errors. Compared with [19], the results from our experiments indicate 34.84% energy gains at the plug in nodes with 512 GB DRAM at 50°C. Out of this reduction, 16.69% is due to the relaxation of the DRAM refresh period, and the rest due to more effective VM consolidation. This energy reduction translates to 29.53% reduced costs for the datacentre operator. Moreover, energy and cost gains are higher in nodes with larger memory capacity, which shows that our approach will scale favorably in future data-dominated datacentres. The main contributions are:

- We introduce an analytic and architecture-agnostic model, which considers the trade-off between the reduction of energy consumption due to reduced margins configuration of DRAMs and the risk of potential SLA violations.
- Based on the model, we introduce Reduced-Margins DRAM (*RM-DRAM*), a novel, cost-driven scheduling and node configuration policy for cloud datacentres. This policy considers DRAM configuration at reduced operating points, on top of cost-aware VM

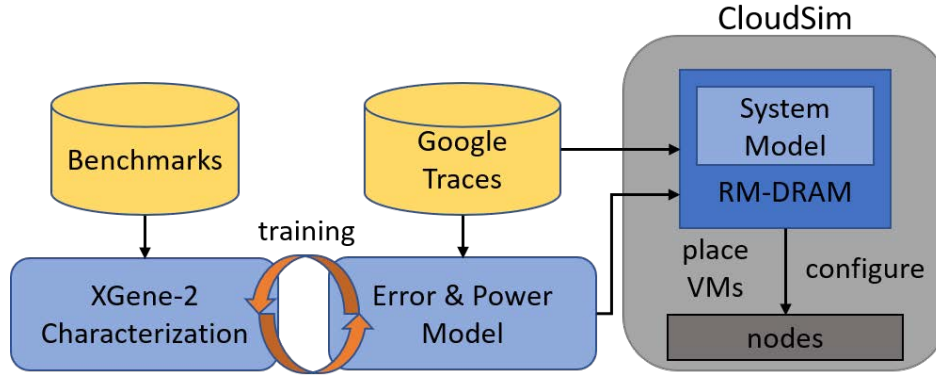


Figure 6.1: Overview of the proposed approach.

consolidation.

- We evaluate our approach in Cloudsim using Google traces.
- We derive a realistic node profile to drive the simulations by characterizing and modelling the workload aware reliability and power effects of the DRAM operation at reduced points on an ARM-based, 64-bit Ampere Computing X-Gene2 server system. The model uses features available in Google traces (such as CPU utilization, memory footprint and memory access rate).

Figure 6.1 illustrates the main elements of our approach. First, we used some benchmarks to characterize the XGene-2 machine and train the error and power models. Then, we created a system model that estimates the cost for the cloud providers and considers the trade-off between energy cost and potential SLA violation penalties. Finally, we designed *RM-DRAM* which utilizes the system model and takes as an input the Google traces and the output of the error and power models and produces the output of VM placing on the nodes and node configurations.

6.1 Problem Modeling

This section introduces an analytic model for the estimation of the datacentre operating cost with DRAM operation at reduced margins. This model is focused on driving decisions for the configuration of DRAMs. It captures key server configuration and VM scheduling aspects, while making realistic assumptions that simplify the problem formulation without loss of generality. Despite some parts of the model bearing similarities with the one discussed

in Section 4.1 for the operation of the CPU at reduced margins, we still present in this section the entire model for completeness. Table 6.1 summarizes the variables we use in the models.

Table 6.1: Summary of variables.

Variable	Description
$CE_{i,p}$	The total number of CEs that occur on n_i at p .
$CE_{i,p,m}$	The total number of CEs for VM_m at p .
$CPU_{m,k}^{alloc}$	The allocated CPU capacity of VM_m at the k^{th} timeslot.
CPU^{CE}	The CPU capacity required for logging and correcting one CE.
$CPU_{i,p}^{CE}$	The CPU capacity required for logging and correcting CEs that occur on n_i at p .
$CPU_{m,p}^{req}$	The requested CPU capacity of VM_m at p .
CPU_m^{SLA}	The requested CPU capacity of VM_m defined in SLA.
$Cost_{i,p}^{W_{i,p}}$	The cost of n_i at p .
$Energy_{i,p}$	The energy cost of n_i at p .
k	The k^{th} timeslot.
K	The number of timeslots of a scheduling period.
$Load_{i,k}$	The load of n_i node at the k^{th} timeslot.
M	The number of the VMs.
MEM_i	The memory amount of n_i .
MEM_m^{SLA}	The requested memory of VM_m defined in SLA.
N	The number of nodes.
n_i	The i^{th} node.
$overhead$	The required amount of time for logging and correcting a CE.
$P_{fail_{RP_{i,p}}}$	The probability of n_i crash due to UEs at p .
p	The scheduling period.
$Price_m$	The VM_m price per time unit.
$P_{i,m,p}^{UE}$	The probability of VM_m to suffer an UE when running on n_i at p .
q_m	A multiplier that increases the severity of SLA penalty for VM_m .
$RP_{i,p}$	The refresh period of the DRAMs of n_i at p .
Res_m	The number of timeslots required for VM_m for rescheduling and restart.
$slotT$	The duration of each timeslot.

$SLAV_{i,m,k}$	The SLA violation cost for the k^{th} timeslot for VM_m on n_i .
$SLAV_{i,p}^{nofail}$	The SLA violation cost for an entire period for n_i without a failure.
$SLAV_{i,p}^{fail}$	The SLA violation cost for an entire period for n_i for a failure.
VM_m	The m^{th} VM.
$u_{i,k}$	The utilization of n_i at k^{th} timeslot.
$UnitPrice$	The energy price per time unit.
$W_{i,p}$	The set of VMs of n_i at p .
$W_{i,p}^{left}$	The set of VMs that left n_i or terminated at p .
$W_{i,p}^{new}$	The set of VMs that arrived at n_i at p .

6.1.1 Nodes and DRAM operating points

We assume a datacentre with N server nodes $n_i, 1 \leq i \leq N$. All nodes have the same CPU and DRAM capacity, denoted by CPU and Mem , respectively. The DRAM of each node n_i operates at refresh period RP_i . By default, this is set to a predefined nominal value, which is the same for all nodes.

It is possible to increase the DRAM refresh period beyond the nominal value to exploit the available cell retention margins. This can reduce power consumption, at the risk of introducing errors to the system. The impact of increased refresh periods in terms of system power consumption and errors is discussed in Section 6.3.

6.1.2 Workload – VMs

Each job is packaged as a VM (the notion of a VM serves as an abstraction for isolated/-contained execution). Let there be M different VMs $VM_m, 1 \leq m \leq M$, which are submitted for execution in the datacentre at various points in time. Each VM has different peak resource requirements (we focus on CPU and memory), declared in a service level agreement (SLA) to be respected by the provider.

Let Mem_m^{SLA} be the peak memory requirement of VM_m specified in the SLA. We assume that the requested memory is allocated in full before a VM starts running and remains allocated to the VM during its entire execution. While hypervisors typically apply transparent page sharing to a limited extent in order to facilitate memory overcommit, in this model, we

assume that memory is not shared among VMs. This assumption simplifies resource management and error accounting, without affecting the validity of the model.

Similarly, let CPU_m^{SLA} be the upper bound for the CPU capacity that will be requested by VM_m , also defined in the SLA. This is expressed as the percentage of the full CPU capacity of the node. CPU requirements are, however, characterized by high volatility. Let $CPU_m^{req} \leq CPU_m^{SLA}$ be the CPU capacity that is requested by VM_m at runtime, again expressed as a fraction of the full CPU capacity.

6.1.3 VM scheduling

The scheduling/placement of VMs on nodes is performed periodically. Similarly to previous chapters, each period p has a fixed length, divided to K timeslots of $slotT$ time units each. For simplicity, we assume that new VMs are accepted and existing VMs terminate at the boundaries of scheduling periods. We use a binary matrix L to encode whether VM_m is live during period p (VM_m was submitted for execution before p and has not yet terminated). If VM_m is live then $L[m, p] = 1$, otherwise $L[m, p] = 0$.

Between periods, the scheduler decides and applies (a) the placement of new VMs and rescheduling of existing VMs to other nodes, and (b) the refresh period $RP_{i,p}$ for the DRAMs of each node n_i for the next period p .

The assignment of VMs to nodes for the period p is encoded in the binary hosting matrix H , where $H[i, m, p] = 1$ if n_i hosts VM_m during p , else $H[i, m, p] = 0$. A node n_i must have enough memory to accommodate the VMs assigned to it: $Mem_i \geq \sum_{m=1}^M H[i, m, p] \times Mem_m^{SLA}$. We assume that the number of nodes is sufficient to accommodate all live VMs at any point in time (datacentre has ample capacity) and that every live VM is placed on some node, in other words $L[m, p] = 1 \Leftrightarrow \exists n_i : H[i, m, p] = 1$. Given that each VM is assigned to a single node $\sum_{i=1}^N H[i, m, p] = 1 \forall m : L[m, p] = 1$.

Apart from placing new VMs on nodes, the scheduler may decide to reallocate existing VMs to different nodes. We assume that the applications running in the VMs are stateless. Thus, VMs can be simply restarted on a different node, without needing to record, transfer and restore state information. We assume the startup time Res_m of a restarted VM_m to be a function of its memory footprint Mem_m^{SLA} and that $1 \leq Res_m \ll K$, i.e., startup takes at least one timeslot but is much shorter than the scheduling period.

We use matrix X to capture the VM reallocation and restart overhead, where $X[i, m, p, k] =$

1 if VM_m actually runs on n_i in the k^{th} timeslot of period p , otherwise $X[i, m, p, k] = 0$. If VM_m is scheduled for reallocation to n_i in p , then $X[i, m, p, k] = 0, 0 \leq k \leq Res_m - 1$ where VM_m is still in its startup phase, while $X[i, m, p, k] = 1, Res_m \leq k < K$.

Finally, we define the workload of node n_i during period p as $W_{i,p} = \{VM_m | H[i, m, p] = 1\}$. This is the set of all VMs that are scheduled for execution on the node for that period. The workload may change between periods, as follows

$$W_{i,p} = W_{i,p-1} - W_{i,p}^{left} + W_{i,p}^{new} \quad (6.1)$$

where $W_{i,p}^{left}$ is the set of VMs that have terminated in the previous period or are selected to be reallocated to another node, and $W_{i,p}^{new}$ are the VMs placed on the node in this period due to new VM arrivals or rescheduling of existing VMs.

6.1.4 Errors due to relaxed refresh rate of DRAMs

Server DRAMs are equipped with error checking and correction (ECC) circuitry. Depending on the algorithm used, ECCs can detect-and-correct up to a certain number of bit errors (detected correctable errors - CE) and detect-only up to an additional number of errors per memory word (detected uncorrectable errors - UEs). In the following, we explain how we incorporate the effects of CEs and UEs in our model.

Correctable errors (correction/logging overhead)

CEs are not fatal for node operation and VM execution. Still, their logging and correction take away a portion of the CPU capacity, which effectively increases the overall system load.

We calculate the CPU capacity required for the logging and correction of CEs that occur in n_i during p as

$$CPU_{i,p}^{CE} = \frac{overhead}{K \times slotT} \times CE_{i,p} \times CPU^{CE} \quad (6.2)$$

where *overhead* and CPU^{CE} are the required amount of time and CPU capacity for reporting and respectively correcting a CE, and $CE_{i,p}$ is the total number of CEs that occur in n_i during p . In turn, we let

$$CE_{i,p} = \sum_{m=1}^M H[i, m, p] \times CE_{i,m,p} \quad (6.3)$$

where $CE_{i,m,p}$ is the expected number of CEs in VM_m during p on n_i when the DRAM refresh period is set to RP_i . Since VMs do not share memory, it is reasonable to assume that CEs occur in different VMs independently.

Uncorrectable errors (node crash failures)

UEs can be detected, but they cannot be corrected. The default behavior in most popular operating systems, such as Linux, is a kernel panic. Hence we assume that UEs lead to a node crash.

We let $P_{i,m,p}^{UE}$ represent the probability of VM_m suffering an UE when running on node n_i during period p . A node will crash as soon as one of the hosted VMs suffers an UE. Given that VMs do not share memory, any VM is a candidate to independently suffer an UE and a single UE is sufficient to crash the node. The probability of n_i failing in period p equals

$$P_{fail_{RP_i,p}} = P \left(\bigcup_{VM_m \in W_{i,p}} UE_{i,m,p} \right) \quad (6.4)$$

Taking a pessimistic approach, we assume that such a node crash always occurs at the beginning of a period and that all VMs that are hosted on the crashed node remain unavailable during the entire period. At the same time, we assume that the VMs of a failed node can be reallocated and restarted in the next period on another node without further delay.

6.1.5 CPU load and CPU capacity allocation to VMs

We estimate the CPU load similarly to Section 4.1.4, but in this work, we also consider the additional load of CEs. To this end, the CPU load of a node in a given timeslot is equal to the total effective requested CPU capacity of all VMs running on it. Moreover, the load increases if CEs occur at the node. Thus the total load of n_i in timeslot k or period p is estimated as

$$Load_{i,k} = \sum_{m=1}^M (X[i, m, p, k] \times CPU_{m,p}^{req}) + CPU_{i,p}^{CE} \quad (6.5)$$

In turn, the CPU capacity allocated to each VM in a given timeslot depends on whether the node is under- or over-loaded:

$$CPU_{m,k}^{alloc} = \begin{cases} X[i, m, p, k] \times CPU_{m,p}^{req} & \text{if } Load_{i,k} \leq 1 \\ X[i, m, p, k] \times (1 - CPU_{i,p}^{CE}) \times \frac{CPU_{m,p}^{req}}{Load_{i,k} - CPU_{i,p}^{CE}} & \text{if } Load_{i,k} > 1 \end{cases} \quad (6.6)$$

If a node is under-loaded, every VM running on it will get the requested CPU capacity. Otherwise, the available CPU capacity, after taking into account the handling of CEs, is shared among VMs proportionally to their requests. Here we assume equal VM priorities, but it is straightforward to adapt the model to a priority-based CPU allocation scheme.

6.1.6 SLA violation cost

A VM may not get the requested CPU capacity at every timeslot. In these cases, the provider pays an SLA violation penalty to the VM owner.

The SLA violation cost is modeled similarly to [11], as in Section 4.1.6 (we explain it again for the completeness of the model). Concretely, we assume it is equal to the product of the normalized extent of the violation (expressed as the difference between the provisioned CPU capacity CPU_m^{SLA} and the one that was actually allocated $CPU_{m,k}^{alloc}$, divided by CPU_m^{SLA}), and the duration of the violation. For any given timeslot k , this is

$$SLAV_{i,m,k} = q_m \times \frac{CPU_m^{SLA} - CPU_{m,k}^{alloc}}{CPU_m^{SLA}} \times SlotT \times Price_m \quad (6.7)$$

where q_m is a multiplier that increases the severity of the penalty according to the importance of VM_m , and $Price_m$ is the price charged to the client for executing VM_m for a time unit, as both agreed in the SLA. Equation 6.7 applies to all sources of SLA violations for VM_m : reallocations to another node, node overloads and node failures.

Based on the above, the SLA violation cost over an entire period for all VMs hosted on a node n_i that does not fail, is

$$SLAV_{i,p}^{nofail} = \sum_{k=1}^K \sum_{m=1}^M X[i, m, p, k] \times SLAV_{i,m,k} \quad (6.8)$$

If n_i fails, none of the VMs will get any CPU capacity, in which case the total SLA violation cost for period p is

$$SLAV_{i,p}^{fail} = K \times SlotT \times \sum_{m=1}^M H[i, m, p] \times q_m \times Price_m \quad (6.9)$$

6.1.7 Energy costs

Let $Power(RP_{i,p}, u)$ be the power consumed by node n_i with a DRAM refresh period $RP_{i,p}$ and CPU utilization u . In turn, the CPU utilization of n_i in timeslot k is given by

$u_{i,k} = \min\{Load_{i,k}, 1\}$. Then, assuming a fixed *UnitPrice* for energy, the total energy cost for n_i over period p is

$$Energy_{i,p} = UnitPrice \times \sum_{k=1}^K Power(RP_{i,p}, u_{i,k}) \times SlotT \quad (6.10)$$

Note that if n_i is not assigned any VMs in p , it will remain idle and its utilization u for that period will be 0. In our experiments, we assume that idle nodes are shut-down. In the same spirit, we assume that failed nodes do not consume any energy during the entire period where the crash failure occurs.

6.1.8 Operating cost

The cost for operating node n_i during period p , assuming it has been assigned workload $W_{i,p}$ and its DRAM is configured to operate with refresh period RP_i , is estimated by

$$Cost_{i,p}^{W_{i,p}} = Pfail_{RP_{i,p}} \times SLAV_{i,p}^{fail} + (1 - Pfail_{RP_{i,p}}) \times (SLAV_{i,p}^{nofail} + Energy_{i,p}) \quad (6.11)$$

More specifically, there are two cases (multiplied by the respective probability). The node may fail due to an UE, in which case the provider will pay the SLA violation cost for all the VMs on the node for the entire period. If the node does not experience an UE (and thus does not fail), the provider may still pay SLA violation costs due to VM reallocation and restart, and/or node overload, plus the cost of the energy consumed by the node to run the VMs at the configured DRAM refresh period.

6.2 Scheduling and System Configuration Policy

Ideally, a scheduling algorithm would find the optimal VM placements and DRAM refresh periods for all nodes and for each period, so as to minimize the total operating cost of a datacentre over all periods. One problem is that, in practice, VM workloads and CPU requirements are not known in advance. Moreover, even if these were known, this optimization problem is NP-hard even for a single period.

In this section, we first introduce some approximations in order to (i) make the problem tractable, and (ii) do so with realistic input. Then, we introduce a workload scheduling and node configuration policy which focuses on maximizing the profit margin for datacentre infrastructure providers by exploiting DRAM operation at reduced margins.

6.2.1 Approximations

In a realistic setting, the future behavior of VMs is not known in advance. As a prediction of VM CPU requirements during the next period, we use a weighted average of CPU occupancy by each VM during previous scheduling periods with exponentially decreasing weights for older observations.

Moreover, we focus on greedily maximizing operator profit at the granularity of each scheduling period, based on Equation 6.11. In a similar vein, before each period, a cost-driven approach is applied to identify the preferred configuration of each node individually rather than at the datacentre-level. This approach improves the scalability of the scheduling policy with respect to the total number of VMs and nodes. The only step requiring a global overview of node status at the datacentre is the allocation of newly arriving and reallocated VMs to nodes, however, this typically involves just a few VMs.

6.2.2 VM Scheduling and Node Configuration – RM-DRAM

We introduce ReducedMargins-DRAM (*RM-DRAM*), a VM scheduling and node configuration policy that tries to reduce the cost for the cloud infrastructure provider by exploiting the refresh margins of DRAMs in an educated manner. In a nutshell, *RM-DRAM* exploits the reduced margins of DRAMs on top of VM consolidation. For VM consolidation, we follow the approach in [19], however we propose cost-driven heuristics. To the best of our knowledge, we are the first to propose a policy that considers DRAM operation at reduced margins on top of VM consolidation.

RM-DRAM is executed at the boundaries of scheduling periods and identifies a good balance between energy cost reduction and the cost of potential SLA violation penalties, by applying a set of heuristics to select a mapping of VMs to nodes and decide the appropriate DRAM refresh period for each node. More specifically, for each scheduling period p , the policy executes a series of steps, described below.

During the first step, equation 6.11 is used to calculate the *a posteriori* optimal RP_i configuration (nominal or reduced) for each node n_i independently, according to the behavior of VMs on that node in the previous period.

For this configuration, node overloads are identified according to the past behavior of VMs, and specific VMs are selected to be allocated to other nodes. Algorithm 4 describes this step in detail. We use the information of past CPU utilization (with exponential aging)

to predict CPU requirements of each VM during the next period. A node is considered to be overloaded if the expected CPU utilization of VMs allocated on that node exceeds a CPU utilization threshold (thr_{util}). Also, overloaded nodes are checked – in parallel – to select the VMs that need to be rescheduled, in a cost-driven manner. A VM is selected to be reallocated to another node and restarted only if this is expected to introduce less cost than risking to leave the node in an overloaded state. The policy selects for reallocation the VM that will introduce the lowest cost (lines 9-14). Finally, VMs selected for reallocation are added to the set of VMs that need to be rescheduled ($VMsForScheduling_p$), and the workload is updated accordingly. This is repeated until no VM reallocation is deemed cost-effective (lines 15-20) or the node in question is no longer considered to be overloaded (line 6).

In the third step, the policy allocates to nodes the VMs that have been selected for reallocation and newly arriving VMs. We sort the VMs in the $VMsForScheduling_p$ set in decreasing CPU utilization order to allocate them to nodes with a Knapsack-like approach. We estimate the effect of allocating the VM at hand on the cost of each node n_i (energy cost and SLA violations cost), assuming the node will maintain the *a posteriori* optimal RP_i calculated during the first step. Each VM in the $VMsForScheduling_p$ is eventually hosted on the node that introduces the lowest cost increment. This step has $\mathcal{O}(\max(m \log m, m \cdot n))$ complexity w.r.t the number m of the VMs that will be hosted on the n nodes.

At this point, the algorithm identifies opportunities to power-off nodes. Starting from the least utilized node, it checks whether all hosted VMs can be allocated to other nodes without overloading them. The destination node for this reallocation is chosen using the heuristic discussed previously with $\mathcal{O}(n \log n)$ complexity where n is the number of the nodes of the datacentre.

The final step, described in Algorithm 5, is to find the optimal configuration independently for each node, given the VM placement for the next scheduling period and the predictions for future CPU requirements of each VM. The optimal node configuration minimizes the cost at the node level, therefore for a fixed VM-to-nodes mapping the total cost for the entire datacentre as well, considering both energy consumption and potential SLA violation costs. Again, we use Equation 6.11 to estimate the cost for each node/memory configuration, and apply the one that introduces the lowest cost. This step is similar to the first step. In the first step, however, we calculate the *a posteriori* optimal configuration of each node, given the known behavior of VMs during the previous period. In this step, we select the operating

Algorithm 4 Overload detection and VM selection for reallocation

Input: $n_i, W_{i,p}$
Output: $W_{i,p}^{left}, VMsForScheduling_p$

```

1: estimate CPU utilization  $u_i$ 
2: if  $u_i > thr_{util}$  then
3:   for  $VM_m$  in  $W_{i,p}$  do
4:      $Cost_m^{realloc} \leftarrow \sum_{k=1}^K SLAV_{i,m,k}^{realloc}$ 
5:   end for
6:   while  $u_i > thr_{util}$  do
7:      $Cost_{min} \leftarrow Cost_i^{W_{i,p}}$ 
8:      $VM^{realloc} \leftarrow NULL$ 
9:     for  $VM_m$  in  $W_{i,p}$  do
10:      if  $Cost_m^{realloc} + Cost_i^{W_{i,p}-\{VM_m\}} < Cost_{min}$  then
11:         $Cost_{min} = Cost_m^{realloc} + Cost_i^{W_{i,p}-\{VM_m\}}$ 
12:         $VM^{realloc} \leftarrow VM_m$ 
13:      end if
14:    end for
15:    if  $VM^{realloc} \neq NULL$  then
16:       $W_{i,p}^{left}$  add  $\{VM^{realloc}\}$ 
17:       $VMsForScheduling_p$  add  $\{VM^{realloc}\}$ 
18:    else
19:      break
20:    end if
21:    re-evaluate  $u_i$ 
22:  end while
23: end if

```

point for the next period, given the (potentially different, due to reallocation and assignment of new VMs) workload of the node for that period, and estimating future CPU requirements of the workload using past information. This step has $\mathcal{O}(1)$ complexity.

Algorithm 5 Node configuration for cost minimization**Input:** $n_i, RPList$ **Output:** Node n_i configuration at $RP_i^{efficient}$

```

1:  $Cost_{min} \leftarrow \infty$ 
2:  $RP_i^{efficient} \leftarrow RP_i^{nominal}$ 
3: for  $RP_i$  in  $RPList$  do
4:   estimate  $Cost_i^{W_{p,i}}$  at  $RP_i$ 
5:   if  $Cost_{min} > Cost_i^{W_{p,i}}$  then
6:      $Cost_{min} \leftarrow Cost_i^{W_{p,i}}$ 
7:      $RP_i^{efficient} \leftarrow RP_i$ 
8:   end if
9: end for
10: configure node  $n_i$  at  $RP_i^{efficient}$ 

```

6.3 Characterization and Modeling of DRAM Power and Error Behavior

This section presents the details of the DRAM error and power models that were developed for estimating $CE_{i,m,p}$, $P_{i,m,p}^{UE}$ and $Power(RP_{i,p}, u_{i,k})$ under various refresh periods in Equations 6.3, 6.4 and 6.10, respectively.

6.3.1 DRAM error modeling

We build a DRAM error model that considers not only the DRAM temperature and the refresh period, as prior works [36,47], but also workload specific features. Such a model allows us to distinguish the DRAM reliability behavior based on the different traces that can be simulated in CloudSim (and which provide information about access rates and memory footprint), while considering recent findings that show that DRAM errors are workload-dependent and may vary by 8x across workloads [51]. Formally, our model predicts a target DRAM error metric (i.e. $CE_{i,m,p}$ or $P_{i,m,p}^{UE}$) for VM_m running on server n_i during scheduling period p , with DRAM operating at refresh period $RP_{i,p}$ and DRAM temperature $TEMP$ as:

$$Metric = ML(Mem_m^R, Mem_m^{SLA}, RP_{i,p}, TEMP) \quad (6.12)$$

where ML is a trained ML model that predicts $Metric$, Mem_m^R is the average memory access rate of VM_m , and Mem_m^{SLA} is the size of memory allocated by VM_m .

The model uses a machine learning technique, the K-nearest neighbors algorithm, which was found to lead to a high DRAM error prediction accuracy [48]. To train the model, we collect $CE_{i,m,p}$ and $P_{i,m,p}^{UE}$ by executing different workloads under various refresh periods and temperatures. For each workload, we collect a set of inherent program features using a profiling tool as in [48]. However, rather than extracting numerous program features, we consider the memory access rate, which was found to have the highest impact on DRAM errors and the memory footprint of each application. Both are available from Google traces.

In our model, we predict $CE_{i,m,p}$ also taking into consideration the DIMM-to-DIMM variation, i.e the difference in DRAM error behavior across different chips. In particular, we use the obtained difference in $CE_{i,m,p}$ across DRAM devices to estimate the standard deviation of $CE_{i,m,p}$. To predict the number of CEs for a specific workload when DRAM operates under relaxed circuit parameters and temperature, we first predict the mean $CE_{i,m,p}$, and then build a Gaussian distribution using this mean and the obtained standard deviation. We randomly sample a $CE_{i,m,p}$ from the obtained Gaussian distribution, thus simulating the DIMM-to-DIMM variation. Figure 6.2 illustrates such a sampling procedure where the mean $CE_{i,m,p}$ (assuming the length $K \times SlotT$ of period p is 5 minutes) is predicted by the model for one of the executed benchmarks, i.e. the *memcached* workload, when DRAM operates under 2.283 s refresh period at 50°C. We extend the model to automatically build the Gaussian distribution of $CE_{i,m,p}$ using the predicted mean and average standard deviation and randomly sample $CE_{i,m,p}$. For example, *sample 1* corresponds to $CE_{i,m,p} = 1800$ (the probability to sample this $CE_{i,m,p}$ is about 0.054), while *sample 2* corresponds to $CE_{i,m,p} = 2200$ (the probability to sample this $CE_{i,m,p}$ is about 0.053). We apply the same approach to consider the DIMM-to-DIMM variation when estimating $P_{i,m,p}^{UE}$.

6.3.2 Characterization Infrastructure

To collect the required datasets and train the model, we run compute intensive benchmarks (Rodinia [84], Parsec [94]), data analytics benchmarks (Ligra) [93] and Cloud workloads, such as memcached [96] on a commodity 64-bit ARMv8-based server, the X-Gene2 Server-on-a-Chip. The server consists of eight 64-bit ARMv8 cores running at 2.4GHz and 4 Micron DDR3 8GB DIMMs operating at 1866 MHz. The server can support up to 256GB of DRAM

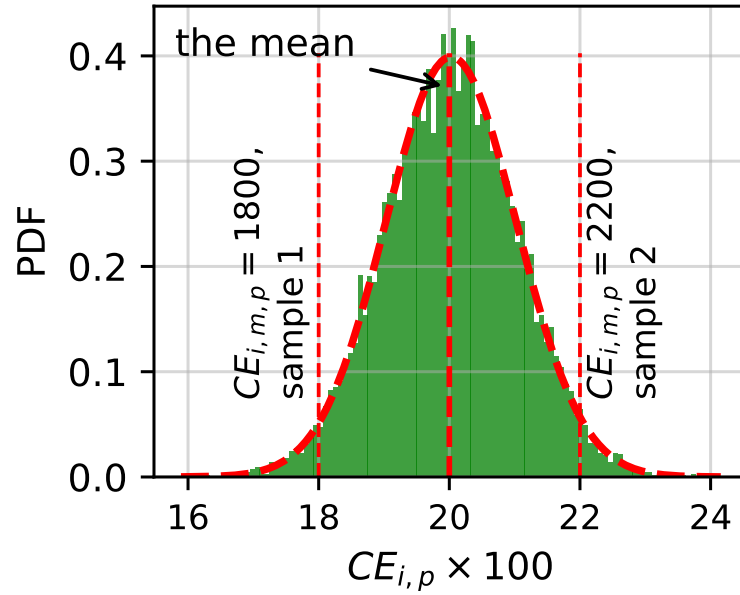


Figure 6.2: Modeling $CE_{i,m,p}$ variation for different DRAM chips.

memory¹. We measure the power of the SoC and DRAM using on-board sensors. The system implements ECC SECDED (single error correction, double error detection) and reports to the Linux kernel all detected CEs and UEs. We quantified the hardware and Linux kernel overhead to handle and report a single CE to be $100 \mu s$ on average. We use this DRAM error handling *overhead* in Equation 6.2.

We perform the experiments under various RP values, ranging from 64ms to 2283ms. At relaxed operating points, we also operate DRAM at the minimum possible supply voltage of 1.428V (lower than the nominal 1.5V). Our study indicated that such a voltage reduces the power with a negligible impact on the number of DRAM errors under any temperature.

To train the DRAM error behavior model for different temperatures, we utilized a temperature-controlled testbed with heating elements, as discussed in a previous study [48]. We use three temperature levels: $50^\circ C$, $60^\circ C$ and $70^\circ C$, that are common in dense server environments [36, 49, 50].

¹In Section 6.4 we extrapolate to 512GB as well, in order to evaluate the effects on big-memory servers.

6.3.3 Modeling DRAM Power and Power at the Plug

DRAM power

DRAM power varies across different workloads. Figure 6.3a shows how the maximum (*Max*) and average DRAM power (*Average*) changes with the refresh period (under the lowest supply voltage). In this figure, *Max* depicts the maximum dynamic power observed in our experiments, which is incurred by the *pagerank* benchmark, while *Average* corresponds to the dynamic DRAM power averaged over all benchmark runs. We observe that the maximum and average DRAM power can be reduced by 21 % and 30 %, respectively, when DIMMs operate under relaxed circuit parameters.

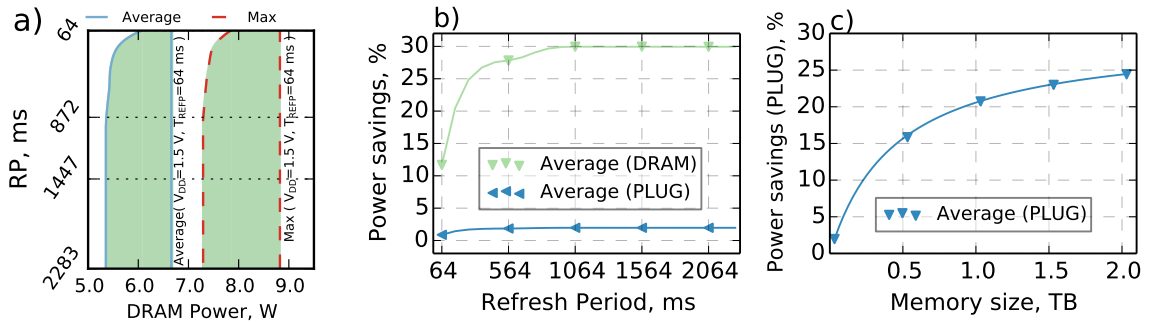


Figure 6.3: a) The maximum and average DRAM power; b) Scaling of DRAM power savings with the refresh period; c) Scaling of DRAM power savings (at the plug) with memory size.

Power at the plug

Figure 6.3b compares the power savings that can be achieved for DRAMs and the entire system i.e. $Power(RP_{i,p}, u_{i,k})$ by scaling the refresh period under the lowered voltage. Power at the plug has been measured using a digital power meter. We see that the maximum savings are up to 3% if we measure power at the plug with a fully utilized CPU. In our experimental framework, we use only 4 DIMMs or 32 GB of memory in total. However, servers in modern datacentres are equipped with many more DRAM devices to process data. The provided power gain that we measure at the plug scales up with memory size and the number of DRAM devices used by servers. Figure 6.3c shows how the power savings scale with memory size, assuming full CPU utilization. To estimate the savings, we project the system power by increasing the number of DIMMs (up to 2TB). Note that the system power grows linearly with the number of DIMM devices [52]. We see that system power savings due to

DRAM operation at reduced margins grow logarithmically with the size of memory (and the number of DIMM devices) and are in the order of 25% for a 2TB system.

The above examples are for the case where the CPU is fully loaded. However, in the general case, the system may be underloaded. The power model in the experimental evaluation also considers CPU utilization.

6.4 Experimental Evaluation

In this section, we present an experimental evaluation of *RM-DRAM* via simulations that quantify the trade-off between energy cost and potential SLA violation penalties. We resort to simulations as it is not feasible to obtain access to a large-scale installation with administrator privileges; the latter is necessary to set the DRAM refresh period beyond the nominal value.

6.4.1 Simulation Setup

For our simulations, we use the popular CloudSim toolkit [25]. CloudSim already supports several useful mechanisms such as VM consolidation, and it allows to configure many parameters of server nodes. We extended CloudSim to support configuration of the nodes at reduced margins by setting the DRAM refresh period and voltage, and to simulate CEs and UEs due to the operation of DRAM at reduced margins, using the memory model discussed in Section 6.3. We also implemented the *RM-DRAM* scheduling policy based on the cost model discussed in Section 6.1.

Similarly to the previous chapters, as an indicative workload, we use a subset of Google traces [26] consisting of 10,000 VMs submitted for execution to the datacentre over 1 day. This corresponds to 288 scheduling periods / invocations of the scheduling policy (the scheduling period is 5 minutes). Each simulation is repeated 10 times and we report the average metrics across all 10 runs.

To evaluate the effect of DRAM operation at reduced margins, we consider the nominal refresh period of 64 ms and four indicative extended refresh period (*RP*) settings of 618, 1173, 1723 and 2283 ms, which can be chosen by our *RM-DRAM* policy to reduce the node's energy consumption. Moreover, we perform simulations with nodes of different DRAM capacity. Higher DRAM capacity may offer more energy gains, however the probability of error increases as nodes accommodate more VMs with a larger memory footprint. We explore

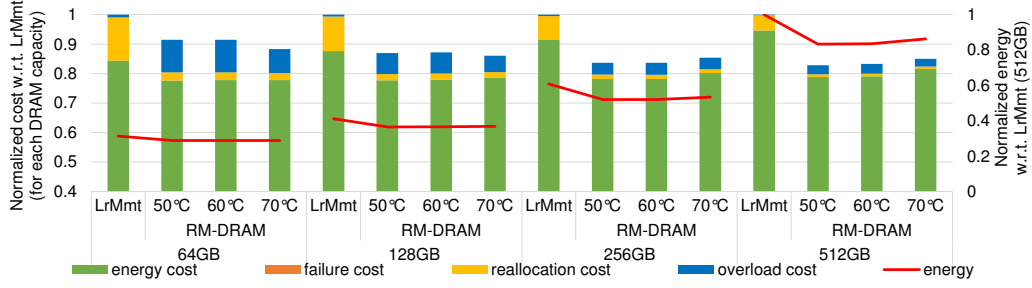


Figure 6.4: Relative cost (left y-axis, bars, normalized w.r.t. $LrMmt$ at the same memory capacity) and relative energy consumption (right y-axis, lines, normalized w.r.t. $LrMmt$ at 512GB), for effectively equal node utilization. Left y-axis starts at 0.4.

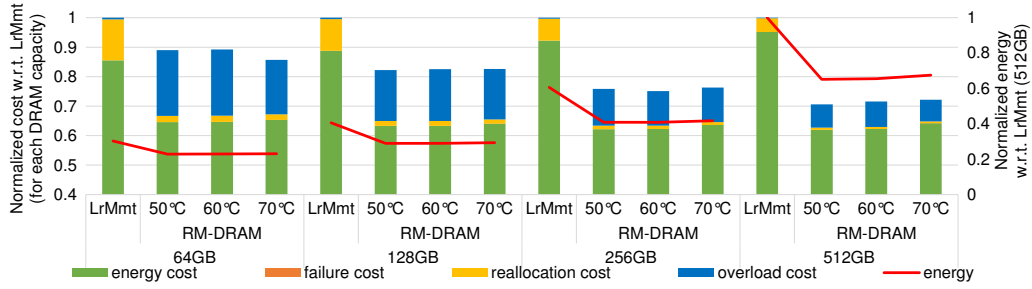


Figure 6.5: Relative cost (left y-axis, bars, normalized w.r.t. $LrMmt$ at the same memory capacity) and relative energy consumption (right y-axis, lines, normalized w.r.t. $LrMmt$ at 512GB), for the same target node utilization. Left y-axis starts at 0.4.

four different datacentre scenarios, where nodes are homogeneous and all have 64, 128, 256 and 512 GB of DRAM, respectively. Our policy receives from the traces the arrival/completion time and maximum CPU requirements CPU_m^{SLA} and memory footprint Mem_m^{SLA} for each VM_m and, for each scheduling period p , the VM's actual CPU requirements $CPU_{m,p}^{req}$ and memory access rate $Mem_{m,p}^R$ (Figure 6.1). The $CPU_{m,p}^{req}$, Mem_m^{SLA} and $Mem_{m,p}^R$ are forwarded to the DRAM error and system power model, together with the evaluated $RP_{i,p}$ setting for each node n_i considered to host VM_m . In turn, the model returns the expected number of CEs $CE_{i,m,p}$ and the probability of UE $P_{i,m,p}^{UE}$ for VM_m .

As discussed in Section 6.3, one more parameter that affects the number of errors is temperature, which is not directly controlled by the operator. Hence, it is important for a cloud policy to be adaptive in this respect. We consider the operating temperatures modeled in section 6.3: 50, 60 and 70°C.

We also use real values for the energy cost [24] and VM pricing [17]. The energy price in our experiments is \$0.15 per KWh and the VM price range for the types of VMs in the traces is \$0.0057 - \$0.4608 per hour, depending on the memory and core requirements of each VM.

In contrast with the previous chapters, we compare our *RM-DRAM* policy with the *LrMmt* policy [19] (rather than [18]) as we want to isolate the benefits of the exploitation of the reduced margins of the DRAMs. *LrMmt* performs VM consolidation and uses local regression for identifying overloaded nodes and selects for migration the VMs with the minimum migration time. *RM-DRAM* assumes stateless VMs and performs reallocation and restart. For fairness, we assume that the migration cost (for *LrMmt*) equals the restart cost (for *RM-DRAM*).

6.4.2 Results and Discussion

Both *RM-DRAM* and *LrMmt* take a target node utilization as a parameter. The goal of this parameter is to guard against node overloads. *LrMmt* blindly tries to respect the target utilization. The cost-aware VM consolidation policy of *RM-DRAM*, however, may violate the threshold – and risk paying SLA violation penalties due to overloads – if this is deemed to be a more cost-efficient option compared to reallocating and restarting excessive VMs and/or activating more nodes. Hence, *RM-DRAM* may end up with higher effective (actual) node utilization and less active nodes. This makes energy gains by DRAM operation at reduced margins hard to distinguish from energy gains due to more cost-effective VM consolidation.

In a first set of experiments, we focus on the gains obtained when DRAM is operated at reduced margins. We compare execution with approximately the same effective node utilization by *LrMmt* and *RM-DRAM* (which do not necessarily correspond to the same target utilization). More specifically, we manipulate the utilization target parameter to achieve an effective utilization of approximately 65% for each policy (a sweetspot between the optimal utilization targets of the two policies). This effectively factors-out the gains due to the more aggressive VM consolidation employed by *RM-DRAM*. Figure 6.4 illustrates the cost reduction and energy gains due to operation at reduced DRAM margins. The bars (left y-axis) correspond to the total cost for the datacentre normalized over *LrMmt* for each memory capacity. The cost is broken down to the cost of energy and the SLA violation penalty cost due to VM reallocations and restarts, node overloads and node failures. The red lines (right y-axis) correspond to the total energy cost of each configuration/policy normalized to the energy cost of *LrMmt* for nodes with 512GB DRAM. Note that one cannot compare the bars of different groups (memory capacities) in a direct way; to do so, one would have to scale their size relatively to the corresponding red line which shows the total cost w.r.t. a common

reference (*LrMmt* for 512GB).

We observe that *RM-DRAM* successfully exploits the reduced margins of DRAMs. The energy gains w.r.t. the nominal DRAM configuration (represented by the green bars on *LrMmt*) depend more on nodes' DRAM capacity, rather than on temperature. They increase with higher DRAM capacity, reaching up to 16.69% for 512GB. In many cases, *RM-DRAM* approaches the highest feasible energy gains for the DRAM capacity at hand. Even in high temperatures, cost gains due to reduced margins are both feasible and significant.

In a second series of experiments, we study the combined effect of DRAM operation at reduced margins and VM consolidation, setting the same target utilization threshold (again 65%) for both applications. Figure 6.5 illustrates the cost benefit (bars) and energy benefit (lines) of *RM-DRAM* w.r.t. *LrMmt* for the four different memory capacity scenarios. We observe a clear trade-off between the cost of reallocating VMs to other nodes and the cost of node overloads. Contrary to *LrMmt*, *RM-DRAM* opts to increase the load of nodes – risking to overload them – thus decreasing both VM reallocations and the number of active nodes. *RM-DRAM* switches off, on average, 26.23% more nodes compared with *LrMmt* and achieves 34.84% energy reduction on average. 16.69% comes from reduced margins and the rest from cost-aware VM consolidation.

Our policy succeeds to reduce the cost for all DRAM capacities and temperatures w.r.t. *LrMmt*. The cost reduction ranges from 11.01% to 29.53% on average, depending on nodes' DRAM capacity and temperature. A larger DRAM capacity offers more opportunities for decreasing the cost thanks to configuration at reduced margins. A higher temperature increases the probability of errors, thus is associated with a higher probability of SLA violation penalty costs. *RM-DRAM* configures nodes more aggressively at reduced DRAM margins when the probability of errors is low, as in the case of 50 and 60°C. However, it risks potential overloads due to error rate deviations related to application- and DIMM-to-DIMM variation. At 70°C, where the expected error rate is higher and there is an increased probability for UEs, *RM-DRAM* is more conservative. As a result, it configures nodes so as to avoid crashes, however at the expense of slightly higher energy and total cost. The average percentage of node crashes in the simulations at 70°C is just 0.0037%. No crashes were observed at lower temperatures.

Figure 6.6 shows the percentage of nodes configured to each of the supported *RP* settings for different DRAM capacities and temperatures. As expected, the most significant parameter

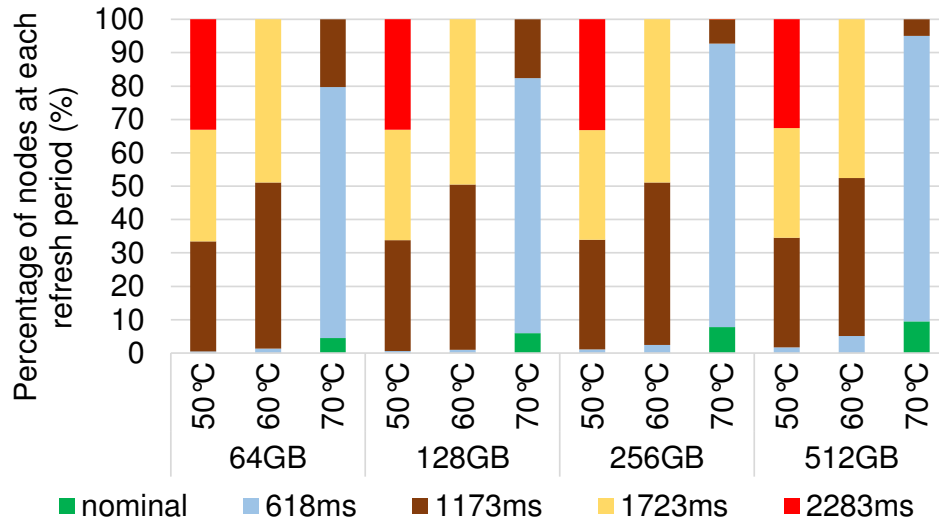


Figure 6.6: Percentage of nodes configured to each refresh period setting (RP) for different DRAM capacities and temperatures.

affecting the selection of RP is temperature. At $50^{\circ}C$ and $60^{\circ}C$ almost all the nodes are configured at reduced margins, however at different refresh rates. At $50^{\circ}C$, the three highest refresh periods are equally preferred. This is expected, as – at $50^{\circ}C$ – they result to almost the same power gains, and very similar average error rates, making the error deviation due to application- and DIMM-to-DIMM-variance a dominant factor in the selection. At $60^{\circ}C$ error rates increase, and $RM-DRAM$ avoids configuration at the highest RP , as it deems that energy gains will not outweigh the penalty costs. At $70^{\circ}C$, where the probability of UEs also increases, $RM-DRAM$ configures some of the nodes at the nominal settings, the other dominant refresh period being 618ms, which is the most conservative of the ones considered in the experimental evaluation.

6.5 Related Work

As discussed in the previous chapters, much of the previous research efforts on energy efficient cloud computing have proposed techniques such as power-aware VM consolidation to pack VMs to as few nodes as possible to enable unused nodes to go to a low-power mode [19]. More recent works introduce optimized VM consolidation algorithms that target the reduction of migrations [28,54]. Also, the support of DVFS [41,42] on modern processors can improve the energy efficiency.

The previous techniques may adversely affect performance and Quality of Service (QoS)

to the extent that they may break SLAs contracted to the Cloud provider. This is further complicated by the dynamically changing workloads and resource requirements. [18] combines DVFS with a VM consolidation policy to reduce power consumption while preventing performance degradation considering the DVFS configuration that would be necessary to maintain Quality of Service. In this work, to isolate the effects due to the exploitation of memory margins and cost-aware migrations (rather than due to CPU techniques for energy efficiency), we compare *RM-DRAM* against [19] (rather than [18]).

Going beyond DVFS, in our previous chapters, we have shown that the conservative CPU voltage margins can be exploited on top of VM consolidation to further improve energy efficiency, by selectively placing VMs on nodes where the CPU is undervolted in a controlled way to save energy without compromising performance. In this work, we focus on DRAMs rather than CPUs. On CPUs, manifested errors usually result to node crashes or SDCs. Memory behavior is more graceful; the probability of errors can be modeled, and CEs manifest at rates that are orders of magnitude higher compared to UEs, for realistic operating temperatures and educated configuration at reduced operating points. This, in turn, calls for a significantly different system model and the design of suitable workload- and system-management policies. Moreover, in this work, we combine DRAM operation at reduced margins with cost-aware, rather than threshold-based VM migration heuristics.

In our evaluation framework rather than using workload unaware DRAM error models as in existing studies [43,44,45,46], we extend our recent workload aware DRAM error model with power values and retrain it to consider features that are highly correlated with DRAM reliability but can also be considered by the relevant input traces in CloudSim.

Chapter 7

Heterogeneity and approximate computing for energy efficiency

In this Chapter, we answer the questions ”Can we apply approximate computing on HPC applications and provide a trustful output?” and ”What are the gains of the combination of heterogeneous and approximate computing for HPC applications?”.

Approximate computing comes to relax some of the computations of the applications to improve the energy efficiency. It is based on the observation that the quality of the output is not equally affected by all the parts of the application [95]. There are several application domains, such as simulations, computer vision, etc. that can benefit from approximate computing, as the relaxation of some of their computations may improve the energy efficiency without sacrificing the quality of the output.

Hardware heterogeneity is another approach to improve performance per Watt. This is a popular path, and the exploitation of heterogeneity and accelerator-based systems (usually GPUs) which combine different architectures, each appropriate for specific computational patterns, can improve the energy efficiency of HPC applications. Moreover, GPUs offer much more computational cores than a typical CPU with a slightly higher power consumption. This makes GPUs superior in terms of performance per Watt, and if an application has a high amount of parallelism and the computational pattern is GPU-friendly, the energy gains may be significant.

Therefore, we investigate if approximate computing can improve the energy efficiency without heavily penalizing the quality of output for HPC applications. On top of that, we combine approximate computing with heterogeneity to enable further energy gains for HPC

Table 7.1: List of benchmark applications and their characteristics.

Program	Domain	Iterative	Input Dependent	Approximation Type	Quality Metric
Smallpt	Scene Renderer	✗	✓	Drop	PSNR
LULESH	Physics simulation	✗	✗	Drop	Relative error
PBPI	Bio-informatics	✗	✗	Drop	Similarity of trees
CG	Numerical solver	✓	✓	Lower accuracy	Relative error

applications. The main contributions of our work are the following:

- To the best of our knowledge, this is the first work that investigates the effectiveness of the combination of heterogeneity and approximate computing in HPC applications considering also the trade-off between energy consumption and quality of the output.
- We evaluate our approach using both large-scale and kernel-scale applications to provide more realistic results, delivering also versions of the application that combine heterogeneity with approximate computing.

7.1 Applications

In this section we provide an overview of applications we used to examine the benefits of heterogeneity and approximate computing, and we briefly discuss the essence of the approximation techniques we applied. Table 7.1 summarizes the applications and the respective application domains. For each application, the first two columns provide information on algorithmic characteristics, namely whether the application is iterative – in the sense of performing steps to converge to a solution, not including applications which perform time steps – and whether its behavior is input – instead of input size – dependent. The last two columns provide the approximation technique(s) and the quality metric used for the evaluation of results.

7.1.1 Smallpt

Smallpt [63] is a global illumination renderer that produces high quality scenes. The input is a plain text description of the scene, containing locations for a variety of objects, light

sources, camera positioning and reflection types of the light (diffuse, specular) and refraction. The application renders the corresponding output scene (image) per pixel.

The core computation uses the path tracing algorithm [64] iteratively. Each iteration recalculates and smoothens the current scene using additional rays. The algorithm updates each pixel of the scene computing an average color value by successively combining the effect of a number of rays in a Monte Carlo fashion.

Approximation: The approximate computation implements the direct lightning method [65], assuming that a ray from a light source travels directly to the destination point. In other words, the approximate version ignores the diffuse reflections, thus it is less computationally intensive compared with the accurate execution which takes into account all types of reflection.

7.1.2 LULESH

Lulesh [66] is a hydrodynamics physics simulation that solves the Sedov blast wave problem for a liquid in the 3D space using Lagrangian hydrocodes. Lulesh exploits the fact that in hydrodynamics, liquids exhibit *boundary conditions* [68], and due to symmetry, it partitions the original domain into sub-domains solving the problem in one sub-domain (one octant of the initial domain). This is the computational region, and Lulesh solves the problem using staggered mesh approximation [67], where the mesh is structured by cubes, and the edges are known as *nodes* and their centroids as *elements*. We have two types of variables: a) the thermodynamic variables such as energy and pressure, and b) kinematic variables such as position and velocity. Lulesh calculates the thermodynamic variables at elements, while the kinematic ones are calculated at nodes. All the variables are initialized to *zero*, except from the position of the nodes and the volume of the elements. Finally, as the blast takes place in the center of the initial domain, Lulesh sets an initial value to the energy of the element that lies closest to the blast.

A common practice in hydrodynamics simulations is to organize the solution of the problem in a sequence of discrete time steps. Lulesh defines an explicit time stepping algorithm and advances to the next time step $t_{n+1} = t_n + \Delta t_n$ to update the variables, where n is the step number and Δt_n is the time increment. Then, Lulesh calculates the force at each mesh node, considering the hourglass filter contribution. Finally, it calculates the new values of the node variables (acceleration, velocity, position) in order to update the element variables (artificial viscosity, pressure, energy).

Approximation: In Lulesh, we introduce two approximations:

- We eliminate the contribution of the hourglass force for the elements that are located far from the blast. This calculation is one of the most time consuming functions, and it is required for the elements that are close to the blast. Our rationale is that dropping this computation for the elements that are far from the blast is not expected to affect solution quality significantly.
- We forward time faster (increase Δt_n), as Lulesh exhibits *energy conservation*.

7.1.3 PBPI

PBPI [69] is a high performance implementation of the Bayesian phylogenetic inference method for DNA sequence data that estimates the likelihood of phylogenetic trees. The application computes the tree that best represents the phylogeny according to the tree likelihood. PBPI starts from random phylogenetic trees where they get accepted or rejected according to their likelihood, given an iterative evolutionary process. At each iteration, the tree with the best likelihood is modified and re-evaluated. PBPI also implements a Bayesian inference algorithm, combined with Monte Carlo Markov Chains (MCMC).

The application uses the Felsenstein's algorithm [70] to estimate the likelihood of a phylogenetic tree. This algorithm traverses the phylogenetic tree in post order to compute the conditional probabilities of the molecular sequence alignment of each internal node, from the corresponding conditional probabilities of its children. These calculations take under consideration the probability of a nucleotide to remain the same (likely) or any mutation (unlikely). These calculations are propagated to the root where the likelihood of the current tree is estimated.

Approximation: PBPI is very sensitive to errors. However, we introduce an approximate version where we drop some calculations of the conditional probabilities of each internal node. We cannot drop the calculation of the probability of a nucleotide to remain the same because it is high, hence we drop the calculations of the probabilities of some mutations. Specifically any of the three mutations may happen to a nucleotide but there is a large difference between the most probable mutation in contrast with the one that is less probable. Therefore we find the mutations that have low probability and drop their calculations.

7.1.4 Conjugate Gradient

Conjugate gradient (CG) is a numerical solver for systems of linear equations. The system has the form $Ax = b$, where A is a square, symmetric and positive-definite matrix, x is the unknown vector and b is a known vector. The algorithm stops when it reaches convergence within a tolerance value, or executes the maximum number of iterations allowed by the user.

CG calculates a new value of the coefficient α in each iteration as shown in equation 7.1. This computation is one of the most time consuming parts of the algorithm. α is used in Equations 7.2 and 7.3, to calculate the new estimation of the solution vector x and the residual vector r (used to evaluate convergence) respectively, where p is one more conjugate vector.

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k} \quad (7.1)$$

$$x_{k+1} = x_k + \alpha_k p_k \quad (7.2)$$

$$r_{k+1} = r_k - \alpha_k A p_k \quad (7.3)$$

Approximation: We implement CG using mixed precision [71]. Computations that are not crucial for the quality of the final solution are implemented using single-precision floating-point, whereas the rest of the algorithm is implemented in double precision. More specifically, we calculate α in single precision floating point, as it is the most time consuming part of the algorithm and – at the same time – the estimation of x is not overly sensitive to the exact value of *alpha*.

7.2 Evaluation

The experimental evaluation was carried out on a dual-socket system equipped with two Intel Xeon E5 2695 processors, clocked at 2.3 GHz, with 128 GB DRAM and an NVIDIA Tesla K80 GPU. The operating system is Ubuntu 14.04 server, using the 3.16 Linux kernel. Applications of our benchmark suite utilize one GPU chip of the K80, therefore the combination of two CPUs results to more power consumption. To calculate the energy consumption on GPU, we measure the power drain with the interval of 2 ms using NVML [72] and multiply the average by the execution time. On CPU, we utilize the Running Average Power Limit (RAPL [73]) counters to measure the energy consumption on each socket.

For each application we provide a spider plot with 5 axes. Applications are executed on CPU and GPU at different degrees of approximation (corresponding to different lines in the

plot). We report the impact of approximations on energy efficiency for CPU and GPU w.r.t. the accurate execution on the same device (*Energy Improv % (CPU)* and *Energy Improv % (GPU)* axis respectively) and on quality w.r.t. the accurate execution (*Quality % (w.r.t. accur.)* axis). Moreover, for each approximation level, we report the energy gain on the GPU w.r.t. the CPU version at the same approximation level (*Energy Gain % (GPU w.r.t. CPU)*), to identify cases where there are device-specific variations in the energy efficiency between the accurate vs. the approximate version. The last axis (*Energy Gain % (GPU w.r.t. CPU accur.)*) reports the energy gain of the GPU approximate execution (again at different degrees of approximation) w.r.t. the accurate execution on the CPU. This metric quantifies the combined energy gains of both heterogeneity and approximate computing over a conventional execution.

As a general observation, we find that in most applications it is possible to improve the energy efficiency both by exploiting heterogeneity and by using approximate computing at the expense of a controlled reduction in the quality of results.

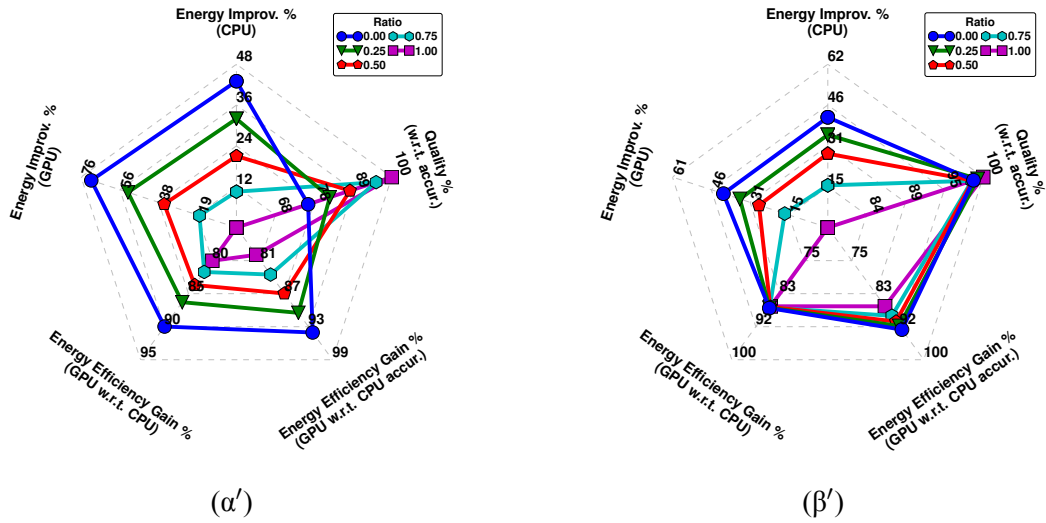


Figure 7.1: (α') Smallpt evaluation, (β') LULESH evaluation

7.2.1 Smallpt

We evaluate Smallpt using the input files included in the original source code, rendering scenes of 1280×1024 pixels for 300 iterations. We evaluate the quality of results using the PSNR of the scene on the last (more refined) iteration of each execution. For illustration purposes we limit the PSNR value of the accurate execution to 48 dB instead of ∞ .

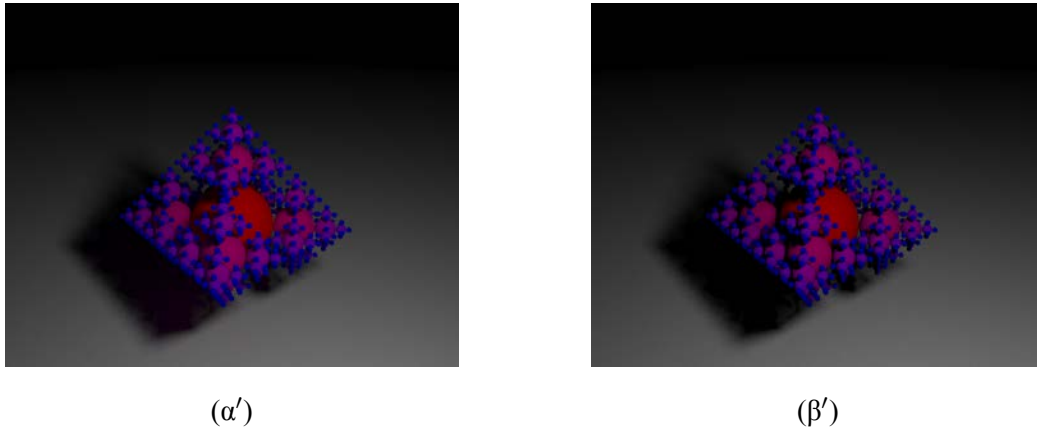


Figure 7.2: (α') Smallpt fully accurate output, (β') Smallpt fully approximate output

The GPU proved to be more energy efficient than the CPU, reducing the energy footprint of the application by about 80.1% for the fully accurate execution, as shown in Figure 7.1 α' . The application has a high degree of GPU-exploitable parallelism: each thread independently calculates the value of one pixel. Apart from heterogeneity, the fully approximate execution had a reduced energy consumption by 43.2% and 71.4% on the CPU and the GPU respectively, compared with the fully accurate execution on the same platform. The approximate version of the code is characterized by less branch divergence than the accurate execution. Therefore, approximation results to steeper energy gains on the GPU. Also, even the fully approximate execution produced scenes of high quality, as depicted in Figures 7.2 β' and 7.2 α' . In all cases the PSNR ranged between 36.8 dB and 45.9 dB.

7.2.2 LULESH

We evaluate Lulesh for a staggered mesh of 1, 000, 000 cubes. As a quality metric we use the relative error of total system energy between accurate and approximate executions.

The error of the fully approximate execution (ratio = 0.00) is 1.3% as shown in Figure 7.1 β' . Elements that are far from the Sedov blast do not significantly affect the spread of the blast wave, hence ignoring them does not introduce significant error. Moreover, the energy conservation property of the system minimizes the error when we increase Δt_n , thus forwarding the simulation more rapidly. The total energy gains from the approximations are 43.6% and 41.6% on the GPU and CPU respectively and they can be attributed mainly to the increased time step. Another interesting observation is that the accurate execution on GPU results to 86.4% less energy consumption than the accurate execution on CPU, due to the fact

that Lulesh exhibits high degree of parallelism and fully utilizes the computational power of the GPU.

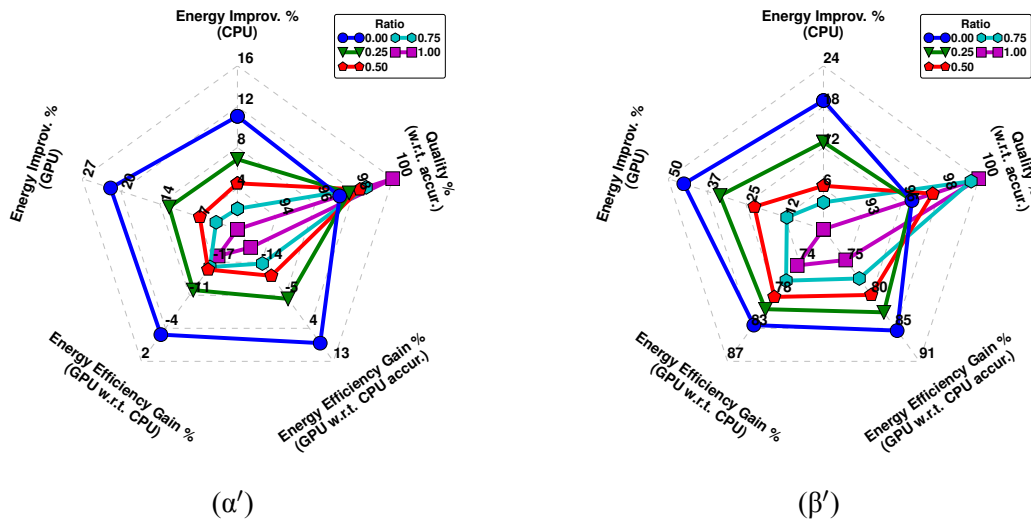


Figure 7.3: (α') PBPI evaluation, (β') CG evaluation

7.2.3 PBPI

We used an input file of 218 organisms, each one of them characterized by a DNA sequence of 10,000 nucleotides. In order to evaluate the quality of our approximate version we used an algorithm for tree comparison that considers both tree topology and branch lengths [74]. We compare the phylogenetic trees produced by the approximate version with those of the accurate and we report their similarity.

Figure 7.3 α' shows that our approximate version can produce trees very similar to those of the accurate version. When we estimate the conditional probability of each internal node, we only eliminate those calculations concerning mutations with very low probability. The CPU proved more energy efficient than the GPU because PBPI exhibits memory access patterns that do not favor GPU. However, the approximate version minimizes this issue on GPU and the energy gain was 22.2% and 11.2% for the GPU and the CPU respectively. As a result, they both consumed almost the same energy for the fully approximate execution.

7.2.4 Conjugate Gradient

For the evaluation of CG we solved linear systems using matrices from the Florida Sparse Matrix Collection [75]. We use the relative error of the approximate solution vs. the solution

computed by the accurate version as a quality metric. We also predefined the total number of the iterations and we used it as stopping criterion.

The relative error of the solution vector x of a fully approximate execution was only 3.8%. We carefully replaced double precision with single precision floating point arithmetic only in parts of the algorithm that are not sensitive to roundoff errors. Figure 7.3β' shows that the energy gain of the fully approximate version is 44.5% and 18.5% for the GPU and the CPU respectively. GPUs are designed to execute single precision computations more efficiently, hence the energy gain due to mixed precision is higher on GPUs. Overall the GPU is more energy efficient than the CPU as it consumes 74.0% less energy, even on fully accurate executions.

7.3 Related Work

To the best of our knowledge this work is the first study of applications that exploit both heterogeneity and approximate computing investigating the trade-off between energy efficiency and quality of output for adaptable levels of approximation. Our work differentiates from existing benchmark suites and studies in several ways.

AxBench [76] investigates the behavior of approximations on both CPU and GPU, but it provides separate implementations (CUDA and C/C++). It exploits two generic approximation techniques: a) Neural Processing Units [78], and b) loop perforation [77]. These are generic techniques, and contrary to this work, we exploit the characteristics of the applications to apply individual approximations on each application. Moreover, we provide OpenCL implementations that are capable to run on a CPU and GPU.

ApproxIt [79] is an approximate computing framework that manages to improve the energy efficiency of iterative applications adjusting the approximation mode at each iteration dynamically. In contrast with our work, it does not consider heterogeneity and it is only used for iterative applications.

EnerJ [80] maps the approximate computations to low-power storage and operations using type qualifiers. More specifically, it isolates the precise computation components from the approximate ones, and it allows the programmer to control the flow from approximate data to precise data. In our work, we do not isolate approximate from accurate data, but we use a task-centric approach with a quality constraint that decides the level of approximation.

Moreover, we consider heterogeneity.

OpenDwarfs [81] is an architecture-agnostic heterogeneous benchmark suite written in OpenCL. Shoc [83] is also written in OpenCL. It compares the performance of modern parallel architectures using microkernels, and it compares OpenCL and CUDA implementations. Rodinia [84] is another benchmark suite that is written in OpenCL (providing also CUDA and OpenMP implementations). It focuses on the study of heterogeneous systems exploring various patterns for parallel applications. Our work is energy efficiency- rather than performance-oriented, and we also introduce additional gains due to the combination of approximate with heterogeneous computing.

In overall, in contrast with the previous benchmark suites, we investigate the benefits of heterogeneous and approximate computing for both large-scale and kernel-scale applications. The use of large-scale applications allows us to explore the more realistic effects of heterogeneity and approximate computing.

Chapter 8

Conclusions

In the previous chapters, we introduced techniques for reducing the energy cost for cloud infrastructure providers by (i) exploiting the reduced margins of operating parameters of the CPU and the DRAMs, and (ii) predicting the resource requirements of dynamic workloads. Moreover, we exploited heterogeneity and approximate computing to reduce the energy footprint of HPC applications, studying the trade-off between energy gains and the quality of the output.

As a first step, we performed a theoretical analysis to identify energy-efficient configurations for the CPU in the multiparametric space through modeling. We studied three models which quantify the cost reduction potential for cloud/edge operators by reducing the energy footprint of the infrastructure both with conservative (VFS) and more aggressive (VS, FS) techniques. All these techniques may affect the provided QoS. We considered penalties due to potential violations of the contractually agreed QoS in cost estimation. Our analysis indicates that voltage overscaling is the most effective technique with cost reduction up to 13.31%. Although frequency overscaling does not reduce the cost significantly, it decreases the execution time of jobs and may allow for additional operator profit by executing more jobs within a given time window.

In the previous modeling we assume that nodes are homogeneous in terms of resources, energy consumption and inherent resilience. Moreover, we assume that the energy cost is constant in time and that we have only computationally intensive tasks. However, real-world cloud infrastructure and workloads are heterogeneous in all those aspects. Similarly, the cost of energy often varies in time. Considering all these parameters paves the ground for interesting scheduling and resource management policies in cloud deployments.

As a next step, we investigated whether undervolting of node CPUs can result to increased profit margins for cloud infrastructure providers. We introduced a model which quantifies the trade-off between energy cost reduction and the cost of SLA violation penalties due to transient node failures. We then discussed *XM-VFS*, a novel online VM scheduling and node configuration policy which uses the model to take decisions on node configuration (at nominal DVFS steps or at reduced margins) and VM placement. We experimentally obtained real values for the parameters used in our model from an Intel Xeon-based server and the newest member (X-Gene 3) of the X-Gene family of Ampere Computing ARM-based high performance servers. The benefits of *XM-VFS* have been evaluated using a set of large-scale simulation campaigns. Our policy outperforms by 17.24% in terms of operator cost *LrMmt*, a state-of-the-art solution combining VM consolidation and DVFS.

Following, we combined the exploitation of the reduced voltage margins of the CPU with a policy that calculates the target CPU utilization per node. To this end, we introduced an adaptive VM scheduling and node configuration policy (*ATUC*). Our policy identifies profitable points in the multiparametric space of VM-node mappings and node CPU configurations, using exclusively information collected at execution time. We combined a scalable, architecture-agnostic, statistical model that estimates a target CPU utilization for each node independently, considering VM allocations to the node and runtime information on past CPU usage of the VMs, with VM consolidation, VFS and CPU operation at reduced voltage margins. We evaluated our approach through simulations, using real datacentre workload traces and hardware parameters derived from real Intel- and ARM-based systems. We found that *ATUC* outperforms, in all cases, *LrMmt* and our previous policy, *XM-VFS*, despite the fact that these policies assume *a priori* workload knowledge and use offline characterization to derive an optimal static target utilization for the workload at hand.

Apart from the CPU, we found that the DRAMs also contribute an important portion of the energy consumption, and they also have operating parameters (voltage and *RP*) with wide safety margins, that can be exploited to reduce the energy cost. To this end, we introduced a VM scheduling policy that focuses on reducing the total energy cost and thereby increasing the profit of datacentre providers by (i) operating the DRAM at reduced margins in an educated manner, and (ii) by applying cost-aware VM consolidation. The policy is based on a new architecture-agnostic system model that jointly considers memory and system level metrics. We evaluated the proposed approach using publicly available workload traces, power

consumption values and estimated error rates for reduced margin operation coming from a memory reliability and power model that are trained using data from the characterization of a real, ARM-based server platform. Our results show that significant cost gains can be achieved compared with *LrMmt*, up to 29.53% for large scale systems, even when operated at adverse temperature conditions. Those gains come from both reducing the excessive supply voltage and refresh period margins of DRAMs, and from applying cost- rather than threshold-based criteria for VM consolidation.

Finally, we studied whether approximate computing can reduce the energy footprint of HPC applications. Considering domain expert wisdom may allow energy savings to occur, without uncontrolled degradation of output quality. On top of that, we combined approximate computing with heterogeneity to investigate further energy gains for HPC applications. For our evaluation, we used 4 application from diverse domains, and our results show that significant energy gains may occur due to the combination of heterogeneous and approximate computing.

In our research, we exploited the reduced margins of hardware parameters of the CPU (supply voltage) and the DRAMs (supply voltage and RP) to reduce the cost of cloud infrastructure providers. Even though the reduction of those margins may result to an increased failure probability, we demonstrated that if we control the operating parameters in an educated manner the cost gains can be significant, particularly if combined with more conservative techniques, such as DVFS and VM consolidation. In our work we also considered the workload variations through time, and we introduced efficient approaches to deal with it, exploiting the observation of VM behavior at runtime. Finally, we studied the problem of energy efficiency for HPC applications, and we showed that applying approximations on select parts of the computations, combined with the exploitation of heterogeneity, we can reduce the energy footprint of the applications significantly, without adverse effects to the quality of results.

References

- [1] John T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Gener. Comput. Syst.*, 22(3):303–312, 2006.
- [2] Djawida Dib, Nikos Parlavantzas, and Christine Morin. SLA-based profit optimization in cloud bursting PaaS. In *Proceedings of 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 141–150. IEEE, 2014.
- [3] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, 2009.
- [4] Konstantinos Parasyris, Vassilis Vassiliadis, Christos D. Antonopoulos, Spyros Lalas, and Nikolaos Bellas. Significance-aware program execution on unreliable hardware. *ACM Trans. Archit. Code Optim.*, 14(2):12:1–12:25, 2017.
- [5] Li Tan, Shuaiwen Leon Song, Panruo Wu, Zizhong Chen, Rong Ge, and Darren J Kerbyson. Investigating the interplay between energy efficiency and resilience in high performance computing. In *Proceedings of Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 786–796. IEEE, 2015.
- [6] Shidhartha Das, David Roberts, Seokwoo Lee, Sanjay Pant, David Blaauw, Todd Austin, Krisztián Flautner, and Trevor Mudge. A self-tuning dvs processor using delay-error detection and correction. *IEEE Journal of Solid-State Circuits*, 41(4):792–804, April 2006.
- [7] Aurélien Cavelan, Yves Robert, Hongyang Sun, and Frédéric Vivien. Voltage overscaling algorithms for energy-efficient workflow computations with timing errors. In *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*, pages 27–34. ACM, 2015.

- [8] Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of 2007 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 38–43. ACM, 2007.
- [9] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.
- [10] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 5. ACM, 2011.
- [11] Saurabh Kumar Garg, Srinivasa K. Gopalaiyengar, and Rajkumar Buyya. SLA-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter. In *Proceedings of Algorithms and Architectures for Parallel Processing*, pages 371–384. Springer, 2011.
- [12] Young Choon Lee and Albert Y Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *Proceedings of 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009*, pages 92–99. IEEE, 2009.
- [13] Kyong Hoon Kim, Anton Beloglazov, and Rajkumar Buyya. Power-aware provisioning of cloud resources for real-time services. In *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*, page 1. ACM, 2009.
- [14] David Blaauw, Sudharsan Kalaiselvan, Kevin Lai, Wei-Hsiang Ma, Sanjay Pant, Carlos Tokunaga, Shidhartha Das, and David Bull. Razor II: In situ error detection and correction for PVT and SER Tolerance. In *Proceedings of Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 400–622. IEEE, 2008.
- [15] Selome K Tesfatsion, Eddie Wadbro, and Johan Tordsson. A combined frequency scaling and application elasticity approach for energy-efficient cloud computing. *Sustainable Computing: Informatics and Systems*, 4(4):205–214, 2014.

- [16] George Papadimitriou, Manolis Kaliorakis, Athanasios Chatzidimitriou, Dimitris Giotopoulos, Peter Lawthers, and Shidhartha Das. Harnessing voltage margins for energy efficiency in multicore cpus. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 503–516. ACM, 2017.
- [17] Amazon EC2 Pricing. <https://aws.amazon.com/ec2/pricing/>.
- [18] Patricia Arroba, José M Moya, José L Ayala, and Rajkumar Buyya. Dynamic voltage and frequency scaling-aware dynamic consolidation of virtual machines for energy efficient cloud data centers. *Concurrency and Computation: Practice and Experience*, 29(10):e4067, 2017.
- [19] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.
- [20] John L Henning. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.
- [21] Anys Bacha and Radu Teodorescu. Using ECC feedback to guide voltage speculation in low-voltage processors. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 306–318. IEEE Computer Society, 2014.
- [22] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204. ACM, 2010.
- [23] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafae, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *Proceedings of ACM SIGPLAN Notices*, pages 37–48. ACM, 2012.
- [24] Electricity price statistics. https://ec.europa.eu/eurostat/statistics-explained/index.php/Electricity_price_statistics.
- [25] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing

- environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.
- [26] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, November 2011. Revised 2014-11-17 for version 2.1. Posted at <https://github.com/google/cluster-data>.
- [27] Anton Beloglazov. Planetlab workload traces. <https://github.com/beloglazov/planetlab-workload-traces>.
- [28] Zhibo Cao and Shoubin Dong. An energy-aware heuristic framework for virtual machine consolidation in cloud computing. *The Journal of Supercomputing*, 69(1):429–451, 2014.
- [29] Young Choon Lee and Albert Y Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.
- [30] Bharan Giridhar, Michael Cieslak, Deepankar Duggal, Ronald Dreslinski, Hsing Min Chen, Robert Patti, Betina Hold, Chaitali Chakrabarti, Trevor Mudge, and David Blaauw. Exploring DRAM organizations for energy-efficient and resilient exascale memories. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12. ACM, 2013.
- [31] Ishwar Bhati, Mu-Tien Chang, Zeshan Chishti, Shih-Lien Lu, and Bruce Jacob. DRAM refresh mechanisms, penalties, and trade-offs. *IEEE Transactions on Computers*, 65(1):108–121, 2016.
- [32] Norbert Engbers and Eric Taen. Green Data Net. Report to IT Room INFRA, European Commission. FP7 ICT 2013.6.2;2014, 2016.
- [33] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. Data Center Energy Consumption Modeling: A Survey. *IEEE Communications Surveys & Tutorials*, 18(1):732–794, 2016.
- [34] Takeshi Hamamoto, Soichi Sugiura, and Shizuo Sawada. On the Retention Time Distribution of Dynamic Random Access Memory (DRAM). *IEEE Transactions on Electron Devices*, 45(6):1300–1309, 1998.

- [35] Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. In *Proceedings of 40th Annual International Symposium on Computer Architecture*, pages 60–71. ACM, 2013.
- [36] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. RAIDR: Retention-Aware Intelligent DRAM Refresh. In *Proceedings of 39th Annual International Symposium on Computer Architecture*, pages 1–12. IEEE, 2012.
- [37] James M Kaplan, William Forrest, and Noah Kindler. Revolutionizing data center energy efficiency, 2008.
- [38] Rajkumar Buyya, Christian Vecchiola, and S Thamarai Selvi. *Mastering cloud computing: foundations and applications programming*. Newnes, 2013.
- [39] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.
- [40] Chaima Ghribi, Makhlouf Hadji, and Djamal Zeghlache. Energy Efficient VM Scheduling for Cloud Data Centers: Exact Allocation and Migration Algorithms. In *Proceedings of 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 671–678. IEEE, 2013.
- [41] Gregor von Laszewski, Lizhe Wang, Andrew J. Younge, and Xi He. Power-aware Scheduling of Virtual Machines in DVFS-enabled Clusters. In *Proceedings of IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE, 2009.
- [42] Qingjia Huang, Sen Su, Jian Li, Peng Xu, Kai Shuang, and Xiao Huang. Enhanced energy-efficient scheduling for parallel applications in cloud. In *Proceedings of 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 781–786. IEEE, 2012.
- [43] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. Flicker: Saving DRAM refresh-power through critical data partitioning. *SIGPLAN Not.*, 46(3):213–224, 2011.

- [44] Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, and Onur Mutlu. Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory. In *Proceedings of 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 467–478. IEEE, 2014.
- [45] Xin Li, Michael C. Huang, Kai Shen, and Lingkun Chu. A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility. In *Proceedings of 2010 USENIX Conference on USENIX Annual Technical Conference*, pages 6–6. USENIX Association, 2010.
- [46] Alan Messer, Philippe Bernadat, Guangrui Fu, DeQing Chen, Zoran Dimitrijevic, David Lie, Durga Mannaru, Alma Riska, and Dejan Milojicic. Susceptibility of commodity systems and software to memory soft errors. *IEEE Transactions on Computers*, 53(12):1557–1568, 2004.
- [47] Moinuddin K. Qureshi, Dae-Hyun Kim, Samira Khan, Prashant J. Nair, and Onur Mutlu. AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems. In *Proceedings of 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 427–437. IEEE, 2015.
- [48] Lev Mukhanov, Konstantinos Tovletoglou, Hans Vandierendonck, Dimitrios S. Nikolopoulos, and Georgios Karakonstantis. Workload-Aware DRAM Error Prediction using Machine Learning. In *Proceedings of IEEE International Symposium on Workload Characterization (IISWC)*, pages 106–118. IEEE, 2019.
- [49] Lauri Minas and Brad Ellison. The Problem of Power Consumption in Servers. <http://www.drdobbs.com/the-problem-of-power-consumption-in-server/215800830>.
- [50] Lev Mukhanov, Konstantinos Tovletoglou, Dimitrios S. Nikolopoulos, and Georgios Karakonstantis. DRAM Characterization under Relaxed Refresh Period Considering System Level Effects within a Commodity Server. In *Proceedings of 2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 236–239. IEEE, 2018.

- [51] Lev Mukhanov, Konstantinos Tovletoglou, Dimitrios S. Nikolopoulos, and Georgios Karakonstantis. Characterization of HPC Workloads on an ARMv8 based server under relaxed DRAM refresh and thermal stress. In *Proceedings of 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 230–235. ACM, 2018.
- [52] Micron Technology. MT18JSF1G72AZ-1G9 - 8GB. <https://www.micron.com/products/dram-modules/udimm/part-catalog/mt18jsf1g72az-1g9>.
- [53] Ehsan Arianyan, Hassan Taheri, and Vahid Khoshdel. Novel fuzzy multi objective DVFS-aware consolidation heuristics for energy and SLA efficient resource management in cloud data centers. *Journal of Network and Computer Applications*, 78(C):43–61, 2017.
- [54] Yaohui Chang, Chunhua Gu, Fei Luo, Guisheng Fan, and Wenhao Fu. Energy efficient resource selection and allocation strategy for virtual machine consolidation in cloud datacenters. *IEICE Transactions on Information and Systems*, 101(7):1816–1827, 2018.
- [55] Fahimeh Farahnakian, Pasi Liljeberg, and Juha Plosila. Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning. In *Proceedings of 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 500–507. IEEE, 2014.
- [56] Zhou Zhou, Jemal Abawajy, Morshed Chowdhury, Zhigang Hu, Keqin Li, Hongbing Cheng, Abdulhameed A Alelaiwi, and Fangmin Li. Minimizing SLA violation and power consumption in cloud data centers using adaptive energy-aware algorithms. *Future Generation Computer Systems*, 86(C):836–850, 2018.
- [57] Electric Power Monthly. <https://www.eia.gov/electricity/monthly/>.
- [58] Waheed Iqbal, Matthew N Dailey, David Carrera, and Paul Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, 27(6):871–879, 2011.

- [59] Leili Salimian, Faramarz Safi Esfahani, and Mohammad-Hossein Nadimi-Shahraki. An adaptive fuzzy threshold-based approach for energy and performance efficient consolidation of virtual machines. *Computing*, 98(6):641–660, 2016.
- [60] Rahul Yadav, Weizhe Zhang, Omprakash Kaiwartya, Prabhat Ranjan Singh, Ibrahim A Elgendy, and Yu-Chu Tian. Adaptive Energy-Aware Algorithms for Minimizing Energy Consumption and SLA Violation in Cloud Computing. *IEEE Access*, 6:55923–55936, 2018.
- [61] Wei Liu, Wei Du, Jing Chen, Wei Wang, and GuoSun Zeng. Adaptive energy-efficient scheduling algorithm for parallel tasks on homogeneous clusters. *Journal of Network and Computer Applications*, 41:101–113, 2014.
- [62] John E. Stone, David Gohara, and Guochun Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *IEEE Des. Test*, 12(3):66–73, 2010.
- [63] David Bucciarelli. Smallpt for OpenCL. <http://davibu.interfree.it/openccl/smallptgpu/smallptGPU.html>.
- [64] James T. Kajiya. The rendering equation. In *Proceedings of 13th Annual Conference on Computer Graphics and Interactive Techniques*, pages 143–150. ACM, 1986.
- [65] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte Carlo techniques for direct lighting calculations. *ACM Trans. Graph.*, 15(1):1–36, 1996.
- [66] Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Technical Report LLNL-TR-490254.
- [67] Mark L Wilkins et al. *Calculation of elastic-plastic flow*. University of California, Ernest L. Lawrence Radiation Laboratory, 1963.
- [68] Henk Kaarle Versteeg and Weeratunge Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Education, 2007.
- [69] Xizhou Feng, Kirk W. Cameron, and Duncan A. Buell. PBPI: A high performance implementation of bayesian phylogenetic inference. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, page 75–es. ACM, 2006.

- [70] Joseph Felsenstein. Evolutionary trees from dna sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, 1981.
- [71] Kristján Jónasson. *Applied Parallel and Scientific Computing: 10th International Conference, PARA 2010, Reykjavík, Iceland, June 6-9, 2010, Revised Selected Papers*. Springer Science & Business Media, 2012.
- [72] NVIDIA. NVML API Reference. <http://docs.nvidia.com/deploy/nvml-api/index.html>.
- [73] Intel. Intel 64 and IA-32 Architectures Software Developer Manual. <https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf>.
- [74] Tamara Munzner, François Guimbretière, Serdar Tasiran, Li Zhang, and Yunhong Zhou. Treejuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility. *ACM Trans. Graph.*, 22(3):453–462, 2003.
- [75] Timothy A Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.
- [76] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 449–460. IEEE Computer Society, 2012.
- [77] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pages 124–134. ACM, 2011.
- [78] Amir Yazdanbakhsh, Jongse Park, Hardik Sharma, Pejman Lotfi-Kamran, and Hadi Esmaeilzadeh. Neural acceleration for gpu throughput processors. In *Proceedings of the 48th International Symposium on Microarchitecture*, pages 482–493. ACM, 2015.

- [79] Qian Zhang, Feng Yuan, Rong Ye, and Qiang Xu. Approxit: An approximate computing framework for iterative methods. In *Proceedings of the 51st Annual Design Automation Conference*, pages 97:1–97:6. ACM, 2014.
- [80] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. EnerJ: Approximate data types for safe and general low-power computation. In *Proceedings of ACM SIGPLAN Notices*, pages 164–174. ACM, 2011.
- [81] Konstantinos Krommydas, Wu-chun Feng, Muhsen Owaid, Christos D Antonopoulos, and Nikolaos Bellas. On the characterization of opencl dwarfs on fixed and reconfigurable platforms. In *Proceedings of 2014 IEEE 25th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 153–160. IEEE, 2014.
- [82] Michalis Spyrou, Christos Kalogirou, Christos Konstantas, Panos Koutsovasilis, Manolis Maroudas, Christos D Antonopoulos, and Nikolaos Bellas. Energy minimization on heterogeneous systems through approximate computing. In *Parallel Computing: On the Road to Exascale*, pages 741–752. IOS Press, 2016.
- [83] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S Meredith, Philip C Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S Vetter. The scalable heterogeneous computing (SHOC) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 63–74. ACM, 2010.
- [84] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of International Symposium on Workload Characterization (IISWC)*, pages 44–54. IEEE, 2009.
- [85] Real-Time Five Minute LMPs. https://dataminer2.pjm.com/feed/rt_fivemin_hrl_lmps/definition.
- [86] Kien Le, Ricardo Bianchini, Jingru Zhang, Yogesh Jaluria, Jiandong Meng, and Thu D Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. ACM, 2011.

- [87] Adel Nadjaran Toosi, Chenhao Qu, Marcos Dias de Assunção, and Rajkumar Buyya. Renewable-aware geographical load balancing of web applications for sustainable data centers. *Journal of Network and Computer Applications*, 83(C):155–168, 2017.
- [88] Zhenhua Liu, Minghong Lin, Adam Wierman, Steven H Low, and Lachlan LH Andrew. Greening geographical load balancing. *ACM SIGMETRICS Performance Evaluation Review*, 39(1):193–204, 2011.
- [89] Íñigo Goiri, Kien Le, Md E Haque, Ryan Beauchea, Thu D Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. Greenslot: scheduling energy consumption in green datacenters. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. ACM, 2011.
- [90] Léo Grange, Georges Da Costa, and Patricia Stolf. Green it scheduling for data center powered with renewable energy. *Future Generation Computer Systems*, 86(C):99–120, 2018.
- [91] Asfandiyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. Cutting the electric bill for internet-scale systems. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 123–134. ACM, 2009.
- [92] Muhammad Abdullah Adnan, Ryo Sugihara, and Rajesh K Gupta. Energy efficient geographical load balancing via dynamic deferral of workload. In *Proceedings of Fifth International Conference on Cloud Computing*, pages 188–195. IEEE, 2012.
- [93] Julian Shun and Guy E. Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, page 135–146. ACM, 2013.
- [94] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.
- [95] Vassilis Vassiliadis, Jan Riehme, Jens Deussen, Konstantinos Parasyris, Christos D. Antonopoulos, Nikolaos Bellas, Spyros Lalas, and Uwe Naumann. Towards automatic

- significance analysis for approximate computing. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, page 182–193. ACM, 2016.
- [96] Tapti Palit, Yongming Shen, and Michael Ferdman. Demystifying cloud benchmarking. In *Proceedings of 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 122–132. IEEE, 2016.
- [97] Avita Katal, Susheela Dahiya, and Tanupriya Choudhury. Energy efficiency in cloud computing data centers: a survey on software technologies. *Cluster Computing*, 26(3):1845–1875, 2023.
- [98] Christina Delimitrou, Daniel Sanchez, and Christos Kozyrakis. Tarcil: reconciling scheduling speed and quality in large shared clusters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, page 97–110. ACM, 2015.
- [99] Christina Delimitrou and Christos Kozyrakis. QoS-Aware scheduling in heterogeneous datacenters with paragon. *ACM Trans. Comput. Syst.*, 31(4), 2013.

APPENDICES

Appendix A

Related Publications

- [1] Christos Kalogirou, Panos Koutsovasilis, Manolis Maroudas, Christos D Antonopoulos, Spyros Lalīs, Nikolaos Bellas. Edge and Cloud Provider Cost Minimization by Exploiting Extended Voltage and Frequency Margins. *Parallel Computing is Everywhere*, pages 814-823. IOS Press, 2018.
- [2] Christos Kalogirou, Panos Koutsovasilis, Christos D Antonopoulos, Nikolaos Bellas, Spyros Lalīs, Srikumar Venugopal, Christian Pinto. Exploiting CPU Voltage Margins to Increase the Profit of Cloud Infrastructure Providers. In *Proceedings of 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 302-311. IEEE, 2019.
- [3] Christos Kalogirou, Christos D Antonopoulos, Spyros Lalīs, Nikolaos Bellas. Dynamic Management of CPU Resources Towards Energy Efficient and Profitable Datacentre Operation. In *Proceedings of Workshop on Job Scheduling Strategies for Parallel Processing*, pages 131–151. Springer, 2022.
- [4] Christos Kalogirou, Christos D Antonopoulos, Nikolaos Bellas, Spyros Lalīs, Lev Mukhanov, Georgios Karakonstantis. Increasing the Profit of Cloud Providers through DRAM Operation at Reduced Margins. In *Proceedings of 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, pages 549-558. IEEE, 2020.
- [5] Panos Koutsovasilis, Christos Kalogirou, Christos Konstantas, Manolis Maroudas, Michalis Spyrou, Christos D Antonopoulos. AcHEe: Evaluating approximate computing and heterogeneity for energy efficiency. *Parallel Computing*, 73:52-67, 2018.

- [6] Georgios Karakonstantis, Konstantinos Tovletoglou, Lev Mukhanov, Hans Vandierendonck, Dimitrios S Nikolopoulos, Peter Lawthers, Panos Koutsovasilis, Manolis Maroudas, Christos D Antonopoulos, Christos Kalogirou, Nikos Bellas, Spyros Lalas, Srikumar Venugopal, Arnau Prat-Pérez, Alejandro Lampropoulos, Marios Kleanthous, Andreas Diavastos, Zacharias Hadjilambrou, Panagiota Nikolaou, Yiannakis Sazeides, Pedro Trancoso, George Papadimitriou, Manolis Kaliorakis, Athanasios Chatzidimitriou, Dimitris Gizopoulos, Shidhartha Das. An energy-efficient and error-resilient server ecosystem exceeding conservative scaling limits. In *Proceedings of 2018 Design, Automation & Test in Europe Conference & Exhibition*, pages 1099–1104. IEEE, 2018.

Appendix B

Contribution to Joint Publications

This appendix discusses my contribution to each of the publications, presented in Appendix A.

In 1, I studied the reduction of the energy consumption for edge and cloud providers through 3 different approaches: a) Voltage and Frequency Scaling, b) voltage overscaling, and c) frequency overscaling. For each approach, I created a model which considers the energy cost of the tasks (for different operating points of the processor), the SLA violation penalties due to failures, and the probability of failure. For the evaluation of the approaches, I used real characteristics of a processor and real prices for the energy and the VMs. Finally, I compared each approach with the cost of the highest nominal operating point of the processor, and I identified CPU configurations that can reduce the energy consumption and the total cost. Mr. Manolis Maroudas and Dr. Panos Koutsovasilis contributed to the analysis of the results.

In 2, I introduced an analytic and architecture-agnostic system model to capture the trade-off between the energy cost and potential SLA violation penalties. I also designed a VM scheduling and system configuration policy that drives VM management and node configuration using the system model. For the requirements of this work, Dr. Panos Koutsovasilis characterized two commercially available machines to identify the voltage margins, the failure probability of the nodes at reduced voltage margins configurations and to create the power consumption model of the entire machine and the model for the performance degradation due to frequency scaling. To evaluate my approach, I implemented these models, the system model and the policy in the simulator and compared my policy with a state-of-the-art policy using real values for the energy price and the price of the VMs. I investigated how the target node utilization affects the cost, I identified the optimal one for both architectures, and I

performed experiments with a lower and higher failure probability than the baseline. I also included experiments with different values for the energy price.

In 3, I introduced an analytic and architecture-agnostic model that deals with the trade-off between energy cost and SLA violation penalties and calculates the target CPU utilization of the node, exploiting the past behavior of the VMs. I created a model using regression analysis to calculate the target CPU utilization, considering the CPU requirements correlation among VMs. Moreover, I designed an adaptive VM management and node configuration policy that exploits the adaptive target CPU utilization calculation for VM consolidation with VFS and CPU configuration at reduced voltage margins. For the evaluation of my approach, I implemented the model and the policy in the simulator. Through my experiments, I quantified the sensitivity of my approach to the selection of initial values for the target CPU utilization. I also examined the cost, per period and for the entire simulation, for different initial values of the target CPU utilization. Finally, I compared it with my policy in Chapter 6 and a state-of-the-art policy.

In 4, I created an analytic and architecture-agnostic system model for managing the energy cost and SLA violation penalties. Based on this model, I introduced a scheduling policy that drives VM consolidation and configuration of the nodes at reduced margins for the hardware parameters of the DRAMs. Dr. Mukhanov Lev characterized the behavior of the DRAMs and created a power model that estimates the power consumption at the plug and the error model for the DRAMs. I implemented these two models, the system model and the policy in the simulator. I also performed the experimental evaluation.

Finally, in 5, I studied the effect of the combination of heterogeneity and approximate computing on the energy footprint of HPC applications. I customized 4 applications to enable execution on heterogeneous systems to evaluate the behavior on different architectures, and I designed and implemented approximations to study the trade-off between energy consumption and quality of output. For LULESH, I contributed to the implementation of the approximate version and the modification for executing the application on heterogeneous systems. The experiments for the evaluation of LULESH were performed by Dr. Koutsovasilis Panos.