Θεωρητικό Ερώτημα

Σε αυτό το κομμάτι μας δόθηκαν **6** θέματα για τα οποία εμείς χρησιμοποιήσαμε το **Google Scholar** για να βρούμε επιστημονικά **paper**, να αντλήσουμε πληροφορίες για το καθένα και να συνθέσουμε μία παράγραφο που να εξηγεί τη σημασία του θέματος, τη χρησιμότητα του και μερικές πληροφορίες για αυτό. Προσπαθήσαμε να χρησιμοποιησουμε κυρίως καταξιομένους οίκους (**Elsevier, IEEE**) και ιδιαίτερα πηγές με οσο το δυνατόν **περισσότερα cites** για κάθε θέμα. Στη συνέχεια ακολουθούν τα 6 θέματα και η αντιστοιχη παράγραφος του καθενός :

Task Dependency Graph

A Task Dependency Graph (TDG) is essentially a Directed Acyclic Graph (DAG) that illustrates how a task-oriented application runs. Its nodes represent tasks, while edges depict task dependencies, ensuring a task can't start until tasks linked to it via incoming edges are done. Almost all interesting and valuable problems involve dependencies in their algorithm flow, resulting in intricate TDGs.

During execution, the scheduler assigns ready-to-run tasks to threads. Depending on the scheduler's design, tasks might be halted (preempted) and resumed later. In various tasking environments like OpenMP, there's a distinction between tied and untied tasks. Tied tasks are exclusively executed by the initiating thread and can only be resumed by that same thread if preempted. Conversely, untied tasks can be resumed by any available thread post-preemption. Both task types offer distinct advantages—tied tasks ensure private data guarantees (e.g., data on the stack), while untied tasks provide greater flexibility to the scheduler. This study explores applications encompassing both task types.

We define TDGs by employing a specific set of fundamental measures. The "work" of computation signifies the overall execution time on a single core, which amounts to the cumulative task times. The "depth" of computation, often referred to as "span," represents the total sum of task times along the critical path. This critical path denotes the longest route, measured in task times, from any source node (a node lacking incoming edges) to any target node (a node without outgoing edges).

Granularity in parallel computing

Granularity in parallel computing encompasses two primary aspects. Firstly, in the context of a parallel computer's architecture, it refers to the ratio between the time required for fundamental communication operations and that needed for basic computation operations. Additionally, concerning parallel algorithms, granularity denotes the number of instructions that can be executed concurrently before synchronization becomes necessary.

Moreover, granularity in a parallel algorithm can be characterized as the relationship between the computational workload and communication load within its implementation (expressed as G=Tcomp/Tcomm). This measurement assesses the balance between computation and communication during the algorithm's execution.

Furthermore, granularity is intricately tied to how a problem is decomposed into tasks. The number and size of these tasks define the granularity of the decomposition. When a problem is divided into a large number of small tasks, it's termed fine-grained, whereas a division into a smaller number of larger tasks is labeled as coarse-grained. For instance, the decomposition of matrix-vector multiplication is typically considered fine-grained due to the multitude of tasks, each performing a single dot-product.

Embarrassing parallel workload

Within the realm of parallel computing, certain workloads or problems exhibit a distinct characteristic known as "embarrassingly parallel." This terminology, also referred to as embarrassingly parallelizable, perfectly parallel, delightfully parallel, or pleasingly parallel, describes scenarios where minimal effort is required to divide the problem into parallel tasks. The hallmark of embarrassingly parallel problems lies in their ability to be seamlessly separated into independent components that can execute concurrently. Typically, such situations arise in scientific computing or graphics, albeit infrequently in broader system applications.

Restated without plagiarism: In the domain of parallel computing, there exists a concept known as "embarrassingly parallel," denoting workloads or problems that effortlessly lend themselves to parallelization. These scenarios, alternatively termed embarrassingly parallelizable, perfectly parallel, delightfully parallel, or pleasingly parallel, involve minimal complexity in the division of the problem into parallel tasks. The prevalence of embarrassingly parallel characteristics is notable in scientific computing and graphics applications, although it is less common in broader system contexts.

Amdahl's Law, a foundational principle in multiprocessor utilization, provides insights into the dynamics of embarrassingly parallel problems. These computational challenges, easily divisible into concurrent components, often arise in scientific and graphical domains. Notably, in scenarios where dependencies or communication needs between parallel tasks are minimal, embarrassingly parallel problems find optimal application. Illustratively, the domain of 3D video rendering, particularly in graphics processing units, serves as a common example. In this context, the handling of each frame (using the forward method) or pixel (utilizing ray tracing) occurs independently, devoid of interdependency. Another instance lies in certain forms of password cracking, which, due to their inherent lack of interdependency, can be efficiently distributed across central processing units (CPUs), CPU cores, or clusters.

Network Function Virtualization

Contemporary networks face a burgeoning challenge of ossification and inflexibility due to the proliferation of diverse network functions and proprietary devices, compounded by the increasing prevalence of middle-boxes. In response to this complex landscape, Network Function Virtualization (NFV) emerges as a transformative paradigm, championed by major Telecommunications Operators (TOs) like AT&T, BT, and DT. Recognized by the European Telecommunications Standards Institute (ETSI), NFV aims to untangle network functions from proprietary hardware, paving the way for a resilient, scalable, and elastic ecosystem. Proposed to address the network's ossification, NFV leverages standard IT virtualization technologies to consolidate proprietary hardware-based network functions into standard commercial devices, such as x86 architecture machines. This decoupling not only mitigates the challenges associated with middle-boxes but also holds the promise of substantial benefits. By enabling the creation of a robust and flexible ecosystem, NFV facilitates automation in network management and orchestration. The separation of network functions into software entities, known as Virtual Network Functions (VNFs), empowers the network to adapt dynamically to evolving requirements.In the realm of NFV research, investigations fall into three primary categories.

The first category explores the integration of NFV with other paradigms, such as optical networks, Internet of Things (IoT), and 5G-supported networks. While these studies delve into the potential of NFV, the direct adoption of standard NFV structures without modifications may introduce unforeseen issues in specific network scenarios.

The second category delves into algorithms for key NFV topics, including Virtual Network Function (VNF) placement, scheduling, and migration. Here, both exact algorithms, offering optimal solutions constrained by network scale, and heuristic algorithms, providing near-optimal solutions with shorter execution times, are explored.

The third category focuses on NFV surveys and reviews, with the challenge of presenting a comprehensive overview. Despite some works offering valuable insights, a truly diverse and comprehensive survey remains a critical need, encompassing hardware-to-software shifts and algorithmic aspects in the dynamic landscape of NFV research. As NFV strives to eliminate network ossification, foster automation, and accommodate evolving service requirements, the collective efforts of the research community aim to define and refine the future of network architectures.

Intelligent Fault-Tolerant Mechanisms

In contemporary technological systems, control mechanisms play a pivotal role in meeting elevated performance and safety requirements. While conventional feedback control designs are effective under normal conditions, they may become inadequate or unstable in the presence of malfunctions in actuators, sensors, or other system components. In response to this challenge, innovative approaches to control system design have emerged, centering on the development of fault-tolerant control systems (FTCS). These systems are indispensable for safety-critical applications such as aircraft, spacecraft, nuclear power plants, and chemical processing plants dealing with hazardous materials. The imperative for high reliability, safety, and fault tolerance in these systems is paramount to withstand potential faults while ensuring stability and optimal performance.

Over recent decades, research in Fault Detection and Diagnosis (FDD) has witnessed substantial growth, resulting in diverse techniques and methodologies. This encompasses survey papers and books exploring fault detection and isolation (FDI) or fault detection and identification (FDI) within the framework of FTCS. Reconfigurable fault-tolerant control systems have garnered attention since the early 1980s, particularly in the aerospace industry. Despite significant efforts, the field lacks standardization in terminology and methodology. The integration of FDD with reconfigurable control presents challenges, necessitating further research to systematically address the interaction between FDD and control system reconfiguration.FTCS can be categorized into passive (PFTCS) and active (AFTCS) types. Passive systems are designed to be robust against presumed faults, while active systems proactively reconfigure control actions to sustain stability and performance during faults. AFTCS places emphasis on designing controllers that are easily reconfigured, implementing high-sensitivity FDD schemes, and developing efficient reconfiguration mechanisms. The time constraints associated with FDD and control system reconfiguration pose a critical challenge. A holistic AFTCS structure comprises a reconfigurable controller, FDD scheme, controller reconfiguration mechanism, and command/reference governor.

Workload Modeling

Workload modeling is a pivotal element in the realm of cloud computing, offering a robust foundation for the analysis and simulation of resource management policies. This not only contributes to the enhancement of Quality of Service (QoS) in cloud systems but also empowers researchers to evaluate and refine policies without the need for large-scale and costly deployments.

However, the landscape of cloud computing presents unique challenges to effective workload modeling, including issues such as the overhead associated with the virtualization layer, limited availability of tracelogs for analysis, and the inherent complexity of diverse workloads. In response to these challenges, we propose a novel web application model. This model is specifically crafted to capture the nuanced behavioral patterns exhibited by different user profiles, thereby facilitating comprehensive analysis and simulation of resource utilization in cloud environments.

The significance of this proposed model extends beyond mere performance analysis. It introduces the capability for controlled parameter modifications, facilitates the repetition of evaluation conditions, and accommodates the inclusion of additional features. Furthermore, the simulation of workloads based on realistic scenarios becomes a valuable asset, particularly in overcoming the scarcity of tracelogs within cloud environments—a scarcity often attributed to business confidentiality concerns.

In the dynamic and ever-evolving landscape of cloud data centers, characterized by heterogeneous hardware and intricate workloads, the proposed web application model addresses a critical gap in existing methodologies. It provides a structured approach to characterizing the diverse behavioral patterns exhibited by cloud applications. Additionally, this model supports the development of performance models, offering a predictive tool for anticipating changes in application patterns. This, in turn, enables dynamic scaling of resources to efficiently meet the expected demand—a capability of paramount importance for cloud providers navigating the challenges of rapidly evolving resource requirements from their applications.

Προγραμματιστικό Ερώτημα

Στα πλαίσια του προγραμματιστικού μέρους της εργασίας κληθήκαμε να παράξουμε κώδικα ο οποίος μετράει το **resource utilization (CPU, RAM, Bandwidth)** και το **latency (response time, tail latency)** κατα την εκτέλεση μιας εφαρμογής υπο διαφορετικά **workloads**. Στην περίπτωση μας επιλέξαμε να χρησιμοποιήσουμε το python framework **Flask** με σκοπό να πραγματοποιήσουμε **GET requests** στο **API** του **LastFm** και στη συνέχεια να κάνουμε **performance modeling**. Έχουμε δύο διαφορετικές εκδοχές :

1) Στην πρώτη εκδοχή το **workload** μας έχει **36 batches** και **frequency 1**. Αυτό σημαίνει ότι σε αυτή την εκδοχή του πειράματος στέλνουμε 36 αιτήματα (**GET requests**) μια φορά όλα μαζί.
2) Στη δεύτερη εκδοχή το **workload** έχει **1 batch** και **frequency 36**. Αυτό σημαίνει ότι σε αυτή την εκδοχή του πειράματος στέλνουμε 1 αίτημα κάθε φορά (**GET request**) 36 φορές.

- Στο παρακάτω screenshot παρατηρούμε τα αποτελέσματα που προέκυψαν απο μια δοκιμή των δύο εκδοχών του πειράματος που προαναφέρθηκε :

Αποτελέσματα

```
+--------------+-------------+---------------------+----------------------+-------------+-----------------+----------------+
|   Batch Size |   Frequency |   Avg Response Time |   Avg CPU Utilization |   Avg Memory |   Avg Bandwidth |   Tail Latency |
+==============+=============+=====================+======================+=============+=================+================+
|           36 |           1 |             35.7929 |              2.76944 |     63.5361 |         1195.78 |        52.9515 |
+--------------+-------------+---------------------+----------------------+-------------+-----------------+----------------+
|            1 |          36 |            0.355296 |                  0.7 |        62.1 |          110052 |       0.355296 |
+--------------+-------------+---------------------+----------------------+-------------+-----------------+----------------+
```

Batch Size	Frequency	Avg Response Time	Avg CPU Utilization	Avg Memory	Avg Bandwidth	Tail Latency
36	1	35.7929	2.76944	63.5361	1195.78	52.9515
1	36	0.355296	0.7	62.1	110052	0.355296

Σχολιασμός Αποτελεσμάτων

- Τα συμπεράσματα που προκύπτουν από τα παραπάνω πειράματα είναι τα εξής :

Εκδοχή 1 (Batch Size : 36, Frequency : 1) :

- **Avg Response Time** (Μέσος Χρόνος Απόκρισης): Ο μέσος χρόνος απόκρισης είναι **υψηλός**, με τιμή **35.7929** δευτερόλεπτα, κάτι που ενδέχεται να οφείλεται στον πιο μεγάλο αριθμό batches.

- **Avg CPU Utilization** (Μέση Χρήση CPU): Η μέση χρήση της **CPU** είναι **2.76%**, μια σχετικά **χαμηλή** τιμή.

- **Avg Memory** (Μέση Χρήση Μνήμης): Η μέση χρήση μνήμης είναι **63.5361%**, η οποία είναι μια σχετικά καλή τιμή.

- **Avg Bandwidth** (Μέση Δικτυακή Ευρυεκπομπή): Η μέση δικτυακή ευρυεκπομπή είναι **1195.78 Bps**.

- **Tail Latency** (Χρόνος Ουράς): Ο χρόνος ουράς είναι **υψηλός**, με τιμή **52.9515** δευτερόλεπτα.

Εκδοχή 2 (Batch Size : 1, Frequency : 36) :

- **Avg Response Time** (Μέσος Χρόνος Απόκρισης): Ο μέσος χρόνος απόκρισης είναι **χαμηλός**, με τιμή **0.355296** δευτερόλεπτα, κάτι που οφείλεται στον χαμηλότερο αριθμό batches.

- **Avg CPU Utilization** (Μέση Χρήση CPU): Η μέση χρήση της **CPU** είναι **0.70%**, μια πολύ **χαμηλή** τιμή.

- **Avg Memory** (Μέση Χρήση Μνήμης): Η μέση χρήση μνήμης είναι **62.1%**, παρόμοια με τον πρώτο τύπο workload.

- **Avg Bandwidth** (Μέση Δικτυακή Ευρυεκπομπή): Η μέση δικτυακή ευρυεκπομπή είναι υψηλή, με τιμή **11005.2 Bps**, υποδεικνύοντας υψηλό ρυθμό μεταφοράς δεδομένων.

- **Tail Latency** (Χρόνος Ουράς): Ο χρόνος ουράς είναι **χαμηλός**, με τιμή **0.355296** δευτερόλεπτα.

Σύγκριση

- Ο πρώτος τύπος workload εμφανίζει **υψηλότερους** χρόνους απόκρισης, **μεγαλύτερη** χρήση μνήμης, και **μικρότερο** bandwidth.

- Ο δεύτερος τύπος workload εμφανίζει **χαμηλότερους** χρόνους απόκρισης, **χαμηλή** χρήση CPU, **υψηλό** bandwidth, και **χαμηλότερο** χρόνο ουράς.

- Επομένως μπορούμε να καταλήξουμε οτι ο **πρώτος τύπος** workload μπορεί να είναι **κατάλληλος** για εφαρμογές που απαιτούν **μεγάλα batches δεδομένων**, αλλά με υψηλότερους χρόνους απόκρισης και οτι ο **δεύτερος τύπος** workload φαίνεται να παρέχει καλύτερη απόδοση με χαμηλούς χρόνους απόκρισης και χαμηλή χρήση πόρων, αλλά με υψηλότερο bandwidth.

Βιβλιογραφία

- https://www.researchgate.net/publication/228525647_HOTRiDE_Hierarchical_ordered_task_replanning_in_dynamic_environments

- https://www.researchgate.net/publication/220698388_Evaluation_of_Parallel_Programs_by_Measurement_of_Its_Granularity

- https://www.researchgate.net/publication/236944300_Introduction_to_Parallel_Computing_2nd_Edition

- https://cs.ipm.ac.ir/asoc2016/Resources/Theartofmulticore.pdf

- https://www.sciencedirect.com/science/article/abs/pii/S1389128618300306

- https://www.sciencedirect.com/science/article/abs/pii/S1367578808000345

- https://www.sciencedirect.com/science/article/abs/pii/S004579061500302X