UNIVERSITY OF THESSALY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Temporospatial Organization of Circuits and Tasks over the Cloud.

by

Panagiotis Oikonomou

A thesis submitted in fulfilment of the
requirements for the degree of Ph.D.

Volos, July 2017

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor Prof. Georgios Stamoulis for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me through all the time of my research and the writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study.

Besides my advisor, I would like to thank the rest of my thesis committee: Asst Prof. Aspasia Daskalopoulou and Asst Prof. Elpiniki Papageorgiou for their insightful comments and encouragement which assisted me in widening my research interests.

I am also grateful to the members of my committee for their patience and support in overcoming numerous obstacles I have faced throughout my research. Furthermore, I would like to express my gratitude to the staff of University of Thessaly and especially to the Depts. of Electrical and Computer Engineer and Computer Science where I have found an intimate environment manned by magnificent researchers.

Finally, I would like to thank my family: my mother, my father and my sister for supporting me spiritually throughout writing this thesis and my life in general.

This thesis is dedicated to:

My mother

ΠΕΡΙΛΗΨΗ ΤΗΣ ΔΙΑΤΡΙΒΗΣ

Χρονοχωρική Οργάνωση Διεργασιών και Κυκλωμάτων σε Υπολογιστικό Νέφος

Η ταχεία και συνεχιζόμενη ανάπτυξη των υπολογιστικών συστημάτων έχει αλλάξει τον τρόπο με τον οποίο οι ερευνητές αντιμετωπίζουν τα προβλήματα ελαχιστοποίησης της κλάσης NP-hard. Οι απαιτήσεις των χρηστών για ταχύτερες ηλεκτρονικές συσκευές και πιο ποιοτική εμπειρία στο διαδίκτυο επηρεάζουν όλο και περισσότερο τις σύγχρονες αρχιτεκτονικές υπολογιστών και δικτύων ενώ οδηγεί στην επίλυση νέων προβλημάτων (βελτιστοποίησης). Οι λύσεις σε τέτοια προβλήματα μπορεί να έχουν σημαντικές ομοιότητες και διαφορές σχετικά με τις χρησιμοποιούμενες τεχνικές, καθώς και στο γενικό πλαίσιο στο οποίο έχει τεθεί το πρόβλημα. Για παράδειγμα, το πρόβλημα του IC placement εμφανίζει χωρικές πτυχές, ενώ ο χρονοδρομολόγηση εργασιών σε υπολογστικό νέφος αποτελείται από χρονικά χαρακτηριστικά. Στην παρούσα Διατριβή προτείνουμε και αξιολογούμε αποτελεσματικές εναλλακτικές λύσεις και στις δύο κατευθύνσεις (χωρικές και χρονικές) με στόχο την ελαχιστοποίηση του χρόνου εκτέλεσης, τη βελτίωση των λύσεων και τη ταυτόχρονη μείωση της κατανάλωσης ενέργειας και της διαδικτυακής κίνησης.

ABSTRACT

Temporospatial Organization of Circuits and Tasks over the Cloud

The rapid and on-going spread of computational systems has changed the way that researchers tackle minimization problems of NP-hard class. Users demand on faster electronic devices and better online experience increasingly affects modern computer and network architectures while it triggers new (optimization) problems to be solved. Solutions to such problems may have substantial similarities and differences regarding the techniques which are used as well as the general framework in which the problem is set. For instance, IC placement problem exhibit spatial aspects while Cloud scheduling consists of temporal ones. In this thesis we propose and evaluate efficient alternatives to both directions (spatial and temporal) under the objective to minimize running time, to improve solutions quality and to reduce simultaneously energy consumption and inter-node network traffic, while meeting the quality requirements.

TABLE OF CONTENTS

LIST OF FIGURES

Figure *Page Number*

LIST OF TABLES

Table *Page Number*

1. Introduction

1.1. Objectives and contribution of the thesis

A plethora of classic optimization problems exhibit spatial or temporal properties. An example of the first, is the 2D Knapsack problem [67] where the target is to place rectangles within a fixed sized plain so that no overlaps occur and the total benefit (summation of the benefits of the placed objects) is maximized. Concerning the temporal property, perhaps the 2-processor scheduling problem [63] and its various variants studied in the field of parallel and distributed computing, e.g., [123], [28], are the most prominent representatives in this category. Inspired by the importance of the spatial and temporal aspects rising in optimization problems, but also by modern problem settings of high practical value that came along with the popularity of the Computational Cloud, in this thesis we focus on providing contributions towards both directions (spatial and temporal). Specifically, we provide contributions to the field of standard cell placement in IC circuits [106] and to the field of Cloud scheduling [122].

In standard cell placement, the cells of a circuit (rectangles of equal height but different length) must be placed on a chip area that is split into fixed equally height rows such that: (i) all cells rest within the chip area, (ii) no cells overlap, (iii) cell positions are row aligned and (iv) some target function is optimized, e.g., total wire length [103], congestion [147] etc. It is straightforward that (i)-(iii) impose constraints of spatial nature that can be seen as special variants of the 2D Knapsack constraints. Quite surprisingly, it turns out that depending on the optimization function (iv), the standard cell placement problem might also exhibit a temporal aspect, distinctly linking it to scheduling problems, as discussed in Chapter 3.

Concerning scheduling problems per se, we focused on modern settings involving the Computational Cloud. Rather than assuming generic job tasks or scientific workflows that apply on a rather restricted audience we tackled the problem of scheduling tasks related to video encoding and transcoding [60]. Video encoding refers to the process of compressing an initially raw video sequence using some standard, e.g., H.264 [146], while transcoding refers to producing multiple outputs from an initial (compressed) sequence, that correspond to various quality levels/bitrates and might even involve a change in

standard e.g., from H.264 to VP9 [49] or HEVC [133]. Since the vast majority of Internet traffic is video related [40] and users access it through devices and network connections of various capabilities, the need for efficient video coding and transcoding is becoming of paramount practical importance. This is also manifested by the number of related companies offering video transcoding as a service, e.g., Amazon Elastic Transcoder [11].

1.2. Contributions

The contributions of the thesis can be summarized as follows.

- Concerning the cell placement problem we show its relation to scheduling and provide scheduling heuristics for the case where voltage drop is taken into account.

- We develop fast but efficient legalization heuristics, i.e., heuristics that start from an initial invalid placement and attempt to perform small changes so that spatial constraints are fulfilled. The heuristics developed are shown to achieve comparable solution quality with a state of the art method [128] but at almost two orders of magnitude less running time.

- We provide a fast parallel implementation for a state of the art legalization method [128] that achieves good speedup without affecting solution quality.

- We consider the case of scheduling video transcoding tasks between a network edge and a central datacenter and develop heuristics to maximize edge usage assuming different QoS levels.

- We consider the case of scheduling transcoding tasks among the various datacenters available for a related service. Quite surprisingly the current state of the art even for the most elaborated services, e.g., Amazon Elastic Transcoder [11] involves manual decisions. We develop heuristics that distribute the load so that network usage (among others) is minimized.

1.3. Outline

The rest of the thesis is organized in the following manner.

Chapter 2 provides a comprehensive literature survey concerning Integrated Circuit (IC) placement on Electronic Design Automation (EDA).

Chapter 3 illustrates the IC Placement problem relevance to job scheduling. Specifically, a detailed formulation of the placement problem is given and steps to reduce its inherent difficulty are illustrated. Additionally, we model the resulting problem rigorously and discuss its complexity, we also demonstrate its relation to the job scheduling problem and outline directions for heuristic design and finally, we evaluate the merits of a greedy approach that takes advantage of the formulation. Parts of this chapter appeared in [114].

Chapter 4 examines the evaluation of standalone variations to the basic Tetris algorithm that aim at significantly improving its performance. Furthermore, we introduce combinations of the standalone heuristics while all of them are evaluated with commonly used benchmark circuits. Parts of this chapter appeared in [44].

Chapter 5 presents the procedure of speeding up the legalization algorithms that offer top quality solutions, such as Abacus, using parallelization. Concretely, we propose a lock-free parallelization framework that can be applied over various legalization schemes, also, we implemented and tested the framework over a legalization algorithm (Abacus) known for its solution quality but also slow running time. Parts of this chapter appeared in [112].

Chapter 6 provides a survey concerning Cloud scheduling, especially under the light of video transcoding jobs.

Chapter 7 analyses the case of live video transcoding on Cloud Edges by way of scheduling heuristics that decide on which jobs should be assigned to an edge mini-datacenter and which to a backend datacenter. Through simulation experiments with

different QoS requirements we conclude on the best alternative. Parts of this chapter appeared in [113].

Chapter 8 investigates the problem of scheduling transcoding jobs over a distributed system comprising of processing nodes that are geographically dispersed and might be whole clusters or even separate data centers. In this chapter we propose algorithms to minimize both the inter-node network traffic and the intra-node energy consumption, while meeting the deadlines and quality requirements. Parts of this chapter are submitted to the ACM Symposium on Cloud Computing 2017 (SoCC'17).

Chapter 9 provides a study about combinatorial optimization. This study consists of four well known optimization problems.

Finally, Chapter 10 summarizes the thesis and provides directions for future work.

2. Placement in Electronic Design Automation

2.1. Overview

Integrated circuit (IC) Placement is a major design problem, especially with the continuous growth in the complexity of modern integrated circuits there is an urgent need for fast placers offering good quality results. Placement is an essential step in Electronic Design Automation (EDA) and an important part in the Physical Design procedure. An indicative abstract description of a Placer, as seen in Figure 2-1, could be a machine that takes as an input a net list and a cell specification library and produces the exact location of each cell while minimizing a number of objective functions such as Half Perimeter Wire Length (HPWL), congestion and power consumption to ensure that a circuit meets its performance demands. IC placement is a difficult problem to solve, an ineffective placer usually leads to more wire which affects the performance and the timing of the circuit. Also, if placement is inefficient next tool in the flow, the router, will become unable to connect all wires or to meet timing.

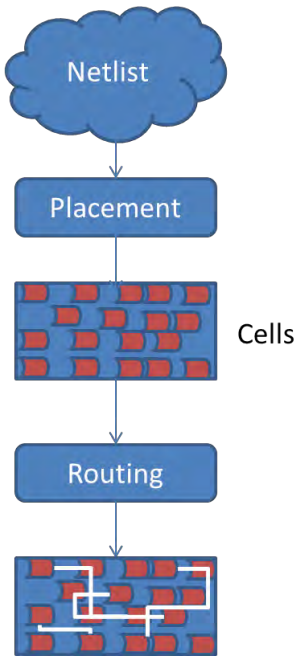


Figure 2-1 IC Placement-Routing workflow

CHAPTER 2. PLACEMENT IN ELECTRONIC DESIGN AUTOMATION

Cell placement has attracted much research interest in the past where a variety of standard cell placers has been proposed by the EDA industry and academia. Placement methodologies can be crudely divided in three major categories based on the way they tackle the problem. The global placement where an initial solution, that contains overlaps and minimizes the cost function, is produced. An intermediate step named legalization that removes all the cell overlaps and finally the detailed placement that refines the final solution to achieve better results.

Global placers solve the problem of placing and spreading cells sufficiently while optimizing the cost function. Various cost functions were considered in the literature such as wire length, routability, time delay, voltage drop and power consumption. Optimization in the global phase is most commonly done without enforcing validity constraints and by incorporating in the model some kind of a repelling force among cells in order to spread them in the chip area. Global placement is the intermediate step between logic synthesis and routing which generates a first and sometimes congested cell distribution in which placement constrains are violated. Such violations include but are not limited to cell overlaps, exceeding critical path delay and net congestion. Global placers can be divided into three major categories: simulated annealing, min-cut partitioning and analytical approaches.

Simulated annealing is a mathematical scheme which can be applied to a number of optimization problems. It starts by calculating a feasible solution and through iterative local changes it computes a better solution. The main drawback of simulated annealing is that it experiences slow convergence rate in large problem set. Timberwolf [125] it was applied as a local optimizer of sub problems. Timberwolf is a probabilistic, iterative improvement technique which approximates the global optimal through iterative cell swaps and moves.

The main idea behind min-cut partitioning is the recursive dividing of the chip area and the design's components, until the regions are small enough to apply a legalization algorithm. Capo [135] is one of the most known placement algorithms that exploit this technique. Dragon2005 [136] and Feng Shui [6] both perform min-cut multi-way

partitioning using hMetis [1] to spread the cells in the chip area, with the former applying simulated annealing to pinpoint the optimal locations

Quadratic and nonlinear techniques constitute the two main subcategories in analytical approaches whereby optimization objectives are modelled by sets of mathematical equations. In Quadratic Optimization the circuit's connectivity is used for the formulation of a quadratic problem and the subsequent minimization of an overall cost function. Examples of this category are: GORDIAN [88], BornPlace [25] and FastPlace [147]. The GORDIAN algorithm is a divide-and-conquer global placement method that is composed of alternating and interacting optimization and partitioning steps that are followed by an optimization of the area utilization. GORDIAN's operation can be divided into three steps, area partitioning, cell partitioning and quadratic calculations. The iterations conclude when there are no sub-partitions left that meet the cell and area constraints we have set (sub-partition dimensions and cell count) while at each iteration a constrain is calculate the center of gravity for the cells belonging to the specific partition. Bonnplace [25] also uses a min-cost flow formulation. The algorithm iteratively augments flows along paths, ensuring that the only flow augmentations that are chosen and applied before the next augmentation step are the ones than can be realized exactly by cell movements. In this way finding the optimal flow isn't guaranteed, nevertheless, the produced solution is always feasible.

In Non-Linear Optimization an approximation of the total wire length is calculated together with cell density using high order models in order to produce better solutions at the expense of runtime. Examples here include: [29], [75], [84], [103] and [110]. White-space reallocation together with force directed placement were used as sub-components of a multilevel optimization approach whereby starting from an initial nonlinear optimization problem that is hard to solve, successive relaxation steps involving cell clustering were applied in an iterative manner. More recent trends include force directed placers whereby cells are spread using a mixture of repelling and attractive forces. Through an iterative procedure equilibrium is reached so that design constraints are satisfied. Two well-known force directed placers are Kraftwerk2 [129] and ePlace [103].

Taking into consideration geometrical constrains in a design, min-cut placers such as [29], [5] and [7] attempt to identify components that should be placed together.

The output of global placement as seen in Figure 2-2 doesn't align cells with rows and might also contain overlaps. Therefore a second legalization step is necessary to meet constraints. An approach is to have as goal the minimum cell displacement between the global placement and the final one (abacus)-(tetris). It is in our best interest to perturb as little as possible each cell's position during legalization, hence the importance of the displacement metric. Contributions in the area of legalization can be classified in two major categories, local and global legalizers.

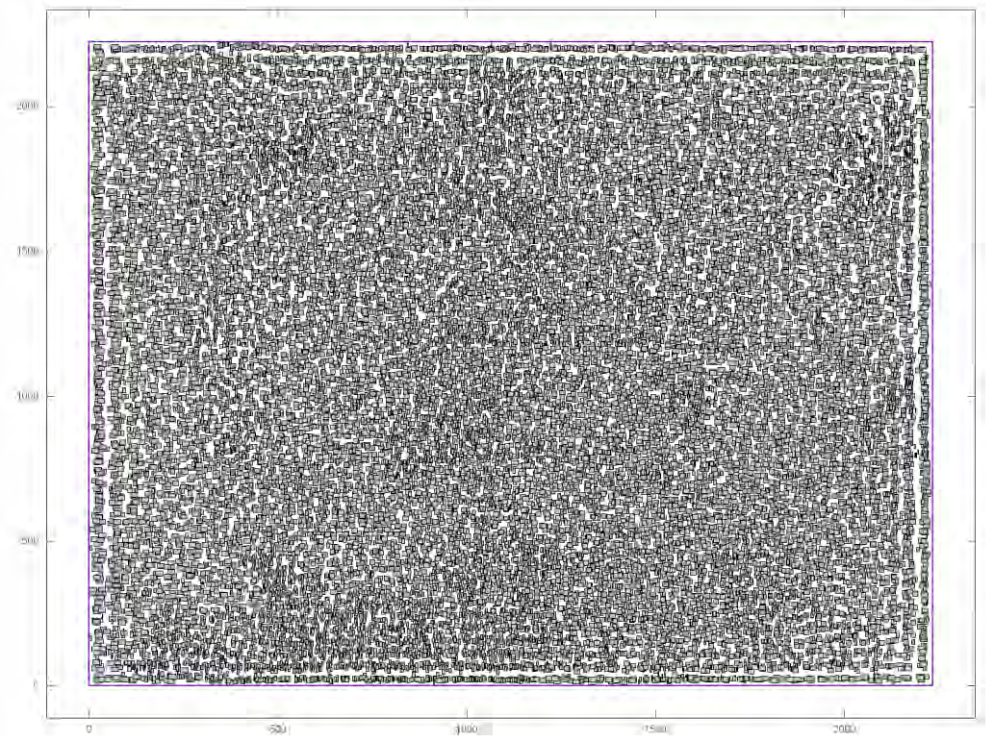


Figure 2-2 Global Placement of ibm05 circuit using NTUplace3

Local approaches achieve legalization by moving cells separately in free spaces or in a ripple motion [77] while the overall congestion is controlled based on white space reallocation [33] or by iterative local refinement algorithms [147]. Some local legalizers

such as [74] and [98] include wire length and displacement minimization using clustering and bin division techniques. Others, such as [114] focus on voltage drop optimization. Dynamic programming is used in [5] and [85] for row fragmentation and placement and in [128] to identify the most promising cell-row assignments. The above methods mainly consider cell placement in a one by one fashion. The simplest and fastest legalization method is the Tetris [73] approach which is used as a yardstick for the performance evaluation of more complex methods that sacrifice running time to achieve better placement. Tetris still remains a popular choice for legalization since it burdens total runtime at a minimum degree, a critical issue for billion cells designs. Abacus [128] also places cells one by one in order of their position on the x-axis. However, in case the vertical alignment of a cell with a candidate row incurs overlaps, a cluster is formed between the cell that is inserted and the ones with which it overlaps and the best position of the cluster is defined through a quadratic formulation. In other terms the key difference of the algorithms is that Abacus might move previously placed cells whereas Tetris doesn't do so. This explains the fact that Abacus achieves lower displacement compared to Tetris but at the cost of increased runtime. In HiBin [98] a bin merged procedure is incorporated, where two different shapes of integrated bins are developed in order to limit the movable scope of each cell. A similar to Abacus approach was followed in [74] with the objective being to minimize cell displacement and HPWL. The presented algorithm uses a row indexing scheme to speed-up the process of finding the best row to insert a cell. In [38] an extension of the clustering method of Abacus is presented that takes into account existing obstacles in the chip area, i.e., preplaced modules that are immoveable. Contrary to Abacus cells are examined in order of their length rather than their position in the x axis as Abacus and Tetris does. In [46] the Abacus legalization scheme was adapted in order to tackle fence regions whereby certain cells must be placed within while others should be excluded. In [118] an Abacus inspired legalization scheme called Jezz is proposed. Jezz considers for row insertion cells, white spaces and blockage nodes in order to cope with obstacles. The resulting scheme was shown to achieve a better performance compared to Abacus but is considerably slower by roughly 20 times.

On the other hand global legalization techniques determine on the positions of multiple cells in a single iteration. Examples of this category are [22], [24], [37] and [53] to

name a few, whereby network flows are used to obtain solution and [42] and [86] that perform cell clustering to reduce search space. Finally, for some restricted problem versions optimal solutions can be found. For instance, for the case of a single row where the assigned cells and their order (left to right) are known, the optimal placement can be found as per [23], while optimal area partitioning is possible in order to offload high density regions as described in [27]. In [37] a history-based legalization scheme is proposed. Min-cost flow formulation is used in order to find a legal placement that presents minimal displacement. Once a viable flow solution is obtained, it is translated to cell movements. During iterations, legalization failures are recorded, and subsequently used in future iterations by a history engine, in order to avoid similar flow realization attempts. In [24] each cell is assigned to a specific region based on its location after the global placement step. A min-cost flow problem is formulated with region boundaries considered as a soft constraint. Dynamic programming is used to decide which cells are going to be moved, therefore realizing the flow.

3. On Formulating and Tackling IC Placement as a Scheduling Problem

3.1. Motivation

Most research on cell placement focused on minimizing the total wire length, hoping to optimize simultaneously the critical path delay as a byproduct. Ideally, a cell placement optimizer should take into account both goals, i.e., wire length and critical path delay offering a set of viable pareto optimal solutions for the designer to choose. One way of tackling the aforementioned two-function optimization problem is to start with one (or more) initial "promising" placements and alter them presumably to gain more towards one or both of the optimization goals. Developing an optimizer with the above characteristics is part of our ongoing work.

In this chapter we present a novel formulation of the cell placement problem with the goal of deriving fast heuristics to output initial placements that have the potential of acting as "good" starting points in a more complex optimization process. Towards this end, we consider optimizing critical path delay in the cell interconnection graph based on individual cell delay characteristics and the premise that placing cells near power sources results in better performance.

Our contributions include the following: (a) we give a detailed formulation of the placement problem and illustrate steps to reduce its inherent difficulty (asymptotic complexity remains the same); (b) we model the resulting problem rigorously and discuss its complexity; (c) we demonstrate its relation to job scheduling problem and outline directions for heuristic design and (d) we evaluate the merits of a greedy approach that takes advantage of the formulation.

3.2. Problem Formulation

3.2.1. Preliminary Definitions

Let C be the set of cells in the circuit we want to place. Cells are the minimal circuit components and can correspond to gates, latches etc. Let c_i denote the i^{th} cell assuming a total order of them. Wired connections exist between cells. We can represent the circuit structure using a graph $G(C, E)$ with vertices depicting cells and edges depicting direct

connections between cells. For most circuit inputs the resulting graph G is undirected and contains circles.

Each c_i is a rectangle of fixed width (let n) and varying length m_i and is associated with a nominal delay value t_i. The chip we wish to design is represented by an $N \times M$ 2D-plane with N showing width and M showing length in the same measurement unit used in n and m_is.

A chip contains a set P of pins. Each pin might be connected with one or more cells. Positions of pins in the plane are assumed to be fixed and part of the input, while pin sizes and delays are considered to be zero. To represent the above, we augment the previous graph forming a new *weighted graph* $G'(V, E')$ with the set of vertices $V = P \cup C$ and E' depicting both cell and pin connections. Each vertex u in the graph is associated with a weight $w(u) = t_i$ if it corresponds to the i^{th} cell, otherwise (it corresponds to a pin) $w(u) = 0$. Henceforth, for simplicity, we will refer to G' as $G(V, E)$, replacing the previous definition of G which is not further needed. We refer to the chip plane together with the pin placement as *Plane_Input* and to the graph G together with cell sizes as *Circuit_Input*.

3.2.2. Generic Problem Statement

Let $p_k = \{u_1, u_2, .., u_k\}$ be a path in G involving k vertices, not necessarily distinct. Path weight $w(p_k)$ is the aggregate weight of vertices in the path, i.e., $w(p_k) = \sum_{i=1}^{k} w(u_k)$. We define the critical (longest) path to be the path p in G of maximum weight. Notice that if G contains circles the longest path weight is infinite. Therefore, we define the critical path in G to be the maximum weight path among the paths satisfying the property that each vertex is visited at most once.

We can now give the generic statement of the placement problem studied in this chapter as follows: *Given Circuit_Input and Plane_Input place the cells in the plane so that: (a) no cell exceeds plane boundaries, (b) no cells overlap and (c) critical path delay is minimized*. Criteria (a) and (b) impose validity constraints, while (c) is the optimization target. In the absence of further details (given in the following subsection), all possible *valid placements* regarding (a)

and (b), result in the same performance concerning (c) which is equivalent to finding the critical path in *G*.

Notice that even by ignoring validity constraints (a) and (b), the problem is still intractable since it is equivalent to finding the longest path in an undirected graph which is known to be NP-hard. On the other hand, we claim (proof omitted) that ignoring the optimization criterion (c) also results in a relevant decision problem (placing all cells so as to satisfy (a) and (b)) that is NP-complete in general, having a knapsack component. Summarizing the above remarks we can state that even with the simplest of the assumptions concerning path delays, the problem discussed in this chapter is hard and requires clever heuristics to master.

As a last remark we would like to mention that in practice, the Knapsack component of the problem is not expected to be the primary challenge. This is because cell sizes do not vary arbitrarily and chip plane usually has significant extra space compared to the total cell area. The methodology presented in section 3.2.4 takes advantage of the above observations.

3.2.3. Time Delay Estimation

The actual delay experienced in a path of the circuit is affected by a plethora of parameters e.g., wire lengths, gate (cell) type etc. Analytical calculations are performed by commercial CAD tools to obtain accurate enough estimations. However, in most cases such calculations can only be performed after the circuit is placed on the chip plane. This is due to the fact that voltage drop depends (aside from other parameters) on the density of placement in a specified region. Given the above remark and the problem's toughness even in the simplest scenario, we decided that adopting sophisticated analytical delay calculations hinters the ability to develop fast and elegant heuristics for the problem, thus, exceeds the scope of this chapter.

Instead, we followed a simple intuitive approach to model the dependencies between cell placement and path delays. Namely, we assume that the "sweetest" spot for placing a particular cell is as close to a power pin as possible. We also assume that the performance

drop is linear to the distance from the power source up to a maximum predefined value *d*. In the presence of multiple power sources we consider the distance from the closest one.

3.2.4. Reducing Problem Complexity

Here, we present the decisions taken in order to reduce the complexity of the placement problem to allow for efficient heuristics.

3.2.4.1. Removing the Knapsack Component

We start by completely removing the knapsack component of the problem namely, criteria (a) and (b). Specifically, from the initial $N \times M$ plane area we only consider for placement reasons the $Rn \times M$ space, where n is the cell width and R an integer such as: $Rn \le N < (R+1)n$. In other words we split the chip plane in rows such that cells fit exactly in every row width-wise. In case the plane cannot be divided exactly in rows of width n, the last row that is a fraction of n in width is discarded. Next, we split the $Rn \times M$ plane in $D = |C|/R$ columns of equal length M/D. Essentially, this splits the whole plane in equally sized slots arranged in R rows and D columns such that their number equals the number of circuit cells C and that a cell can fit in a slot width wise. In general, a circuit cell can be placed anywhere in the plane accounting for increased complexity regarding the three problem criteria. Let s_j denote the j^{th} slot in a total ordering of them. By assuming all cells to be of equal size length-wise and fit exactly to the slot length, i.e., $m_i = M/D \; \forall 1 \le i \le |C|$ the potential placement positions of a cell equal $|C|$, the number of slots. This significantly reduces search space and can be elegantly encoded in a $|C| \times |C|$ placement matrix X of Boolean values, whereby $X_{ij}=1$ iff c_i is placed in slot s_j, 0 otherwise.

```
longestPathCalculation(DAG(G))
M:=Topological sort array of DAG(G);
for all 1 ≤ i ≤ |C| visit[i]:=false; endfor
longestPath:=NINF;
while (exists i: visit[i]=false) do
        s:=min{i: visit[i]=false};
        cost:=calculateCost(s);
        if (cost>longestPath) then longestPath:=cost; endif
endwhile
return longestPath;
```

```
calculateCost(s)
u:=M[s]; Visit[s]:=true; cost:=w(u);
if (u has no outgoing edges in DAG(G)) then
        return cost;
endif
for all v: (u, v) exists in DAG(G)
        temp:= w(u) + calculateCost(v);
        if (temp>cost) cost:=temp; endif
endfor
return cost;
```

Figure 3-1 Pseudocode for longest path calculation in DAG(G).

3.2.4.2. Making Critical Path Calculation Tractable

As mentioned in section 3.2.1 longest path calculation in a weighted undirected graph is NP-hard. Therefore, in order to make critical path calculation tractable, we transform the graph *G* that is part of the *Circuit_Input*, into a directed acyclic graph DAG(*G*) using the following steps. First we run DFS starting from all input pins. The result of this step is a forest of DFS trees in which some of the original edges are missing. Then we perform topological sort and construct DAG(*G*) by taking every undirected edge (*u*, *v*) in *G* and adding a directed edge (*u*, *v*) in DAG(*G*) iff (*u*, *v*) doesn't already exist and *u* appears before *v* in the topological order. Otherwise, in case (*v*, *u*) doesn't exist in DAG(G) and *u* appears after *v* in the topological order, the directed edge (*v*, *u*) is added in DAG(G).

Having constructed DAG(G) in a preprocessing step and calculated its topological order, longest path calculation can be performed in O(|DAG(E)|) time where DAG(E) is the set of edges in DAG(G). Notice that from the DAG construction process |DAG(E)| = $|E|/2$, where E is the set of edges in G. Figure 3-1 shows longest path calculation in pseudocode. As a final note we would like to note that the longest path computed in DAG(G) needs not be equal to the longest path that exists in G (in fact the opposite would imply that P=NP). In constructing DAG(G) we lost some of the information graph G contained. Figure 3-2 illustrates an example. Assuming all vertex weights to equal 1, the initial undirected graph G (Figure 3-2-a) has a longest path of 5 shown in dashed lines. One possible outcome of applying DFS in G starting from the grey vertex, is shown in Figure 3-2-b which depicts the DFS tree in bold lines. Based on this tree the final constructed DAG is given in Figure 3-2-c.As it can be observed the resulting graph has a longest path of 3.

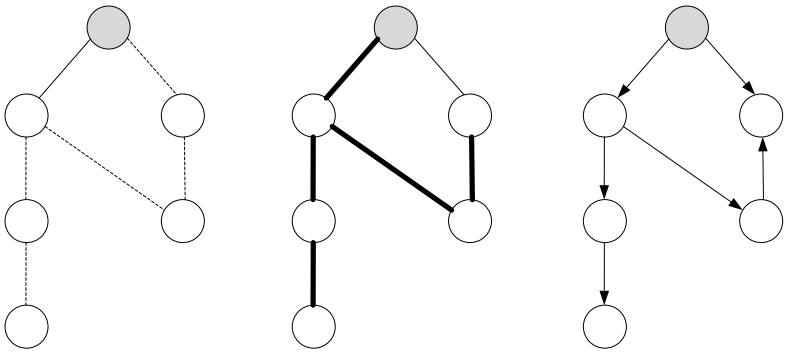


Figure 3-2 An example of DAG construction

3.2.4.3. Slot Speedup Calculations

As mentioned in section 3.2.3 we assume that the time delay introduced in a cell is linear to the distance from the closest power source. Here, we model the interdependence of the placement decision and cell delay as a speedup factor of slots. As it will be discussed in following sections this modeling allows us to view the cell placement problem as a scheduling problem, thus, being able to benefit from the rich

literature that exists in scheduling, e.g., [95] for a somehow old survey. We illustrate the process through the following example.

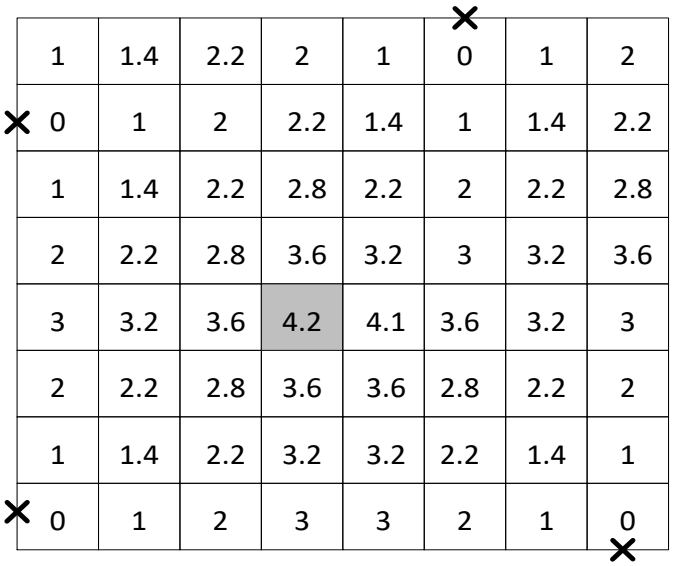


Figure 3-3 An example of distance calculation with 4 sources.

Consider the case where a circuit of 64 cells, each being a perfect square with edge size of 1, must be placed in a chip plane of size 8×8. Figure 3-3 shows the plane arranged in 8×8 slots. Let there be 4 power sources with their places shown as an x mark in Figure 3-3. Slots containing power sources are assumed to have a distance of zero. For the remaining (slot, source) pairs, the Euclidean distance is calculated between the slot's center and the center of the slot containing the power source. In the example, 4 such distances will be calculated per slot (one for each source). Figure 3-3 records for each slot the smallest of the four calculated distances (let $dist_i$ for slot s_i).

Afterwards, for each slot s_i a speedup factor (let f_i) is calculated as follows. The slot with the largest distance $dist_{max}$ (grey slot in the example) is assumed to have a speedup factor of 1 and decrease performance compared to a slot with distance 0 by a factor of $d \in [0,..1]$. This means that the speedup factor of a distance 0 slot (let f_{max}) is set to $1/(1-d)$. The remaining f_i factors (see Figure 3-4) are calculated proportionally as follows:

$$f_j = (1 - d\frac{dist_j}{dist_{\max}})/(1-d) \tag{3.1}$$

1.33	1.29	1.20	1.22	1.33	1.42	1.33	1.22
1.42	1.33	1.22	1.20	1.29	1.33	1.29	1.20
1.33	1.29	1.20	1.14	1.20	1.22	1.20	1.14
1.22	1.20	1.14	1.06	1.10	1.12	1.10	1.06
1.12	1.10	1.06	1	1.01	1.06	1.10	1.12
1.22	1.20	1.14	1.06	1.06	1.14	1.20	1.22
1.33	1.29	1.20	1.10	1.10	1.20	1.29	1.33
1.42	1.33	1.22	1.12	1.12	1.22	1.33	1.42

Figure 3-4 An example of speedup factor calculation.

Notice that Eq. 3.1 implies that $f_j \geq 1 \ \forall j$. Slot speedups depict the potential benefit (delay wise) of placing a cell in a particular slot. In particular, assigning cell c_i to slot s_j leads to a delay: $t_{ij} = t_i / f_j$. Put it in other terms, by placing c_i at s_j, the weight $w(c_i)$ of the corresponding vertex at DAG(G) will become $w(c_i)/f_j$.

3.2.5. The Relaxed Cell Placement Problem

Instead of viewing the cell placement problem as a problem of placing cells in a plane (section 3.2.2), we use the methodology of section 3.2.4 to formulate it as a problem of placing speedup factors (essentially slots) to cells (graph vertices).

Relaxed Cell Placement Problem (RCPP): Given Circuit_Input and Plane_Input, place slot speedup factors f_js in the vertices of DAG(G) so that: (a) each factor is placed at exactly one vertex, (b) each vertex gets exactly one factor and (c) the longest path in DAG(G) is minimized.

Equivalently, using placement matrix X (introduced in Sec. 3.4.1), path cost definition becomes:

$$w(p) = \sum_{u \in p} w(u) / f_j \mid c_i \equiv u \wedge X_{ij} = 1 \tag{3.2}$$

and (RCPP) can be stated more formally as: *Given Circuit_Input and Plane_Input find the values of X so that: (a)* $\sum_{\forall i} X_{ij} = 1, \ \forall j$ *, (b)* $\sum_{\forall j} X_{ij} = 1, \ \forall i$ *and (c)* $w(lp) = \max\{w(p) \mid p \in DAG(G)\}$ *is minimized.*

We would like to point out that a solution to RCPP is not necessarily a valid solution for the statement of section 3.2.2 since a cell might not fit in a single slot. Therefore a legalization step is required, whereby cells are moved in the plane in order to satisfy the validity constraints of section 3.2.2. This can be done using one of the existing techniques in the related literature, e.g., [82]. After legalization, it is likely that a cell will overlap multiple slots, in which case we assume that the speedup factor assigned to the cell equals the lowest speedup of the slots it overlaps. It is also worth mentioning that the legalization step is required not only to restore validity constraints, but also to exploit the optimization potential that exists when cell size is less than slot size (through suitable alterations a cell might be able to move to a better slot in terms of speedup). Due to space limitations we omit a detailed discussion on legalization.

3.2.6. Complexity

RCPP doesn't suffer from the intractability of longest path calculation and bears no Knapsack component as the initial problem statement of section 3.2.2 did. Proving complexity when speedups obey Eq. 3.1 is part of ongoing work. With the assumption that slot speedups can be arbitrary, the relevant decision RCPP can be proved to be NP-complete having a 2-processor scheduling component..

Given a scheduling problem instance, we construct a graph component consisting of two disjoint paths. Let $|p_1(V)|$ be the number of vertices in the first path, $|p_2(V)|$ in the second, then $|p_2(V)|$ is set to $|J|$ - $|p_1(V)|$, i.e., both paths in total have vertices

corresponding to the number of jobs $|J|$ in the scheduling instance. Then for each job we create a speedup factor $T = \prod_{\forall job} time(job)/time(job)$, for a total of $|J|$ factors. By assuming that the initial weight at each vertex equals $\prod_{\forall job} time(job)$ we end up after the assignment of speedups to vertices with the two disjoint paths having vertex weights corresponding to job times. It is then easy to argue on the equivalence between asking for the longest path and minimizing the time of the most loaded processor. However, such equivalence exists only by assuming that jobs are split into processors so that one processor gets $|p_1(V)|$ jobs and the other $|p_2(V)|$. Therefore, in order to be able to calculate the job make-span regardless of the split, we construct all possible distinct graph components ($\lfloor |J|/2 \rfloor$ in number), corresponding to splits: (1, $|J|$-1), (2, $|J|$-2) etc. We then make $\lfloor |J|/2 \rfloor$ copies of the speedup factors used above and suitably combine graph components so that a solution to RCPP gives both the best split and the best job assignment, thus, a solution to scheduling.

3.3. Heuristics

Having identified in section 3.2.5 that RCPP has a scheduling component, we can use it to design efficient heuristics. The intuition behind them is to identify a set of "heavy" paths in DAG(G) and judiciously split the set of available speedups among them. Discussing and evaluating such heuristics is left for an extended version.

Greedy

```
//f: array with speedups sorted in decreasing order
for all 1 ≤ i ≤ |C|
        assignVertex[i]:=false;
        speedupCounter:=1;
endfor
stop:=false;
while (!stop) do
        cost:=longestPathCalculation(DAG(G));
        vSet:={vertices in the above longest path};
        if (exists i such that ci belongs in vSet and
        assignVertex[i]=false) then
              candidate:= i: ti=max{tj for every cj belonging in vSet};
              tcandidate:=tcandidate/f[speedupCounter];
              speedupCounter++; assignVertex[candidate]:=true;
```

```
        else
            stop:=true;
        endif
endwhile
return cost;
```

Figure 3-5 Pseudocode for the greedy heuristic.

Here, we evaluate a *greedy heuristic* that consists of calculating the longest path and assigning the largest speedup factor to its heaviest vertex. The process is repeated iteratively until no further improvement can be obtained (i.e., the vertices of the calculated longest path have already speedup assignments). Figure 3-5 shows the pseudocode of the algorithm.

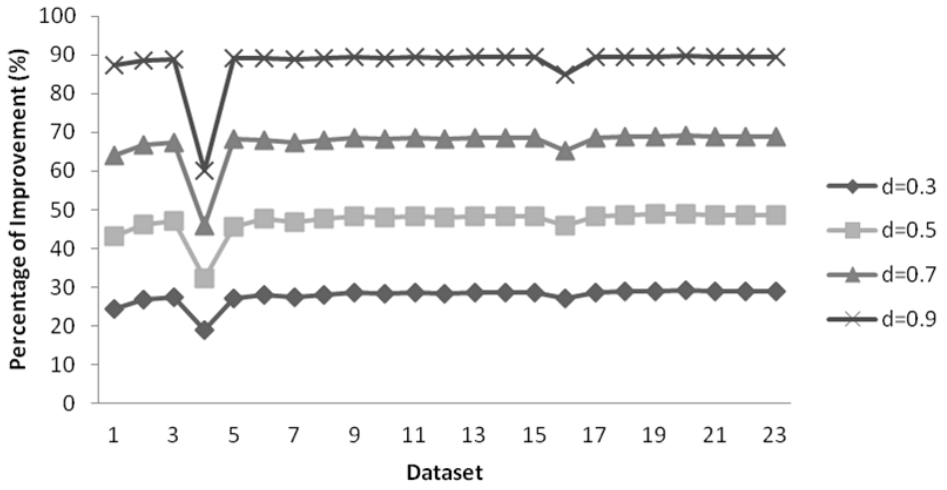


Figure 3-6 Performance improvement of Greedy vs. Initial longest path (d=0.3, 0.5, 0.7, 0.9).

3.4. Experiments

This section presents our experimental findings. It is organized as follows. Section 3.4.1 illustrates the experimental setup. Section 3.4.2 and section 3.4.3 illustrate the results under two comparison scenarios. Section 3.4.4 gives the runtime performance. Finally, section 3.4.5 includes a small discussion.

3.4.1. Experimental Setup

We used 23 circuits from the ISCAS'89 benchmark. We performed experiments with the parameter *d* taking the following values: 0.3, 0.5, 0.7 and 0.9. Recall that *d* depicts how bad the worst slot placement is, compared to the best one. We consider the following placement alternatives:

Greedy. The output of the algorithm presented in section 3.3.

Random. Randomly assigns cells to slots. We record the best result obtained from 100 different runs.

Initial. This is the initial longest path that exists in DAG(G) before assigning any speedup factor.

In all the experiments we record performance as a percentage of improvement compared to a base placement. The smallest dataset is dataset_1 consisting of 19 cells and the largest dataset_20 consisting of 741 cells.

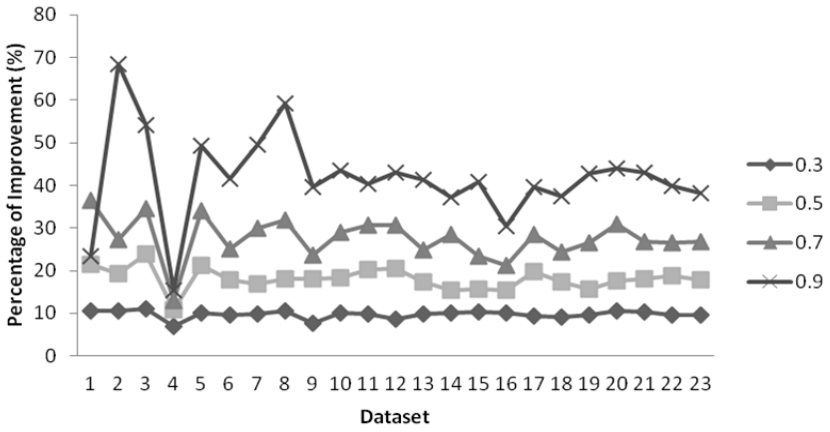


Figure 3-7 Performance improvement of Greedy vs. Random (d=0.3, 0.5, 0.7, 0.9).

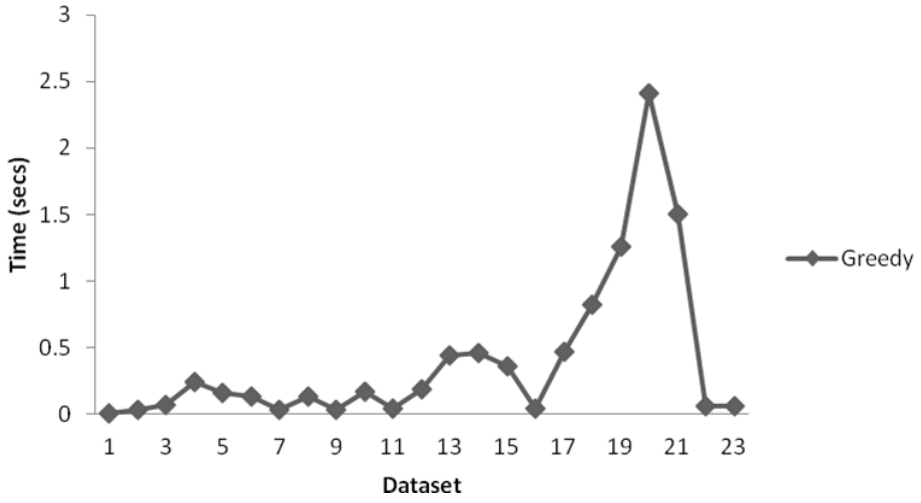


Figure 3-8 Runtime of Greedy (secs).

3.4.2. Greedy vs. Initial Critical Path

Here we compare the performance of the greedy heuristic versus the longest path that exists before any speedup operator is assigned. Figure 3-6 shows the percentage of improvement by the Greedy algorithm compared to the initial. As expected considerable performance improvements are achieved by Greedy. Furthermore, the potential optimization gains increase to the slot speedup difference (in d=0.3 improvement is between 20-30%, while for the extreme case of d=0.9 improvement is close to 90% in the majority of the cases).

3.4.3. Greedy vs. Random

Next, we compare Greedy against Random. Figure 3-7 presents the results for all circuit datasets as the percentage of improvement of the Greedy heuristic compared to the Random. Greedy is a clear winner with performance improvement varying from 10% up to 70% depending on the case. This result is particularly encouraging since it means that even when slot speedup differences are small random cell placement at slots is clearly inferior compared to the greedy approach.

3.4.4. Execution Time

Figure 3-8 illustrates the runtime of Greedy algorithm. Results were obtained using a laptop carrying an Intel Core i5 processor running in 2.4 GHz with 6GB memory. It can be noted that the runtime never exceeded a couple of seconds.

3.4.5. Discussion

Summarizing the experimental results we can say that: The Greedy heuristic is fast enough and the optimization margin using our formulation on its dataset appears to be of significance. Finally, Greedy outperforms Random.

4. Heuristics for IC Legalization

4.1. Motivation

Cell placement is the problem of placing the cells of a VLSI circuit over a chip area, such that no cell exceeds area boundaries, no two cells overlap and cells are aligned into the rows that the chip area is split into. In standard cell placement, all cells have the same width (potentiallly different lengths) and the chip area is split into equally sized rows. Most commonly, the final cell placement is defined using a two step approach. First, a *global placer* spreads the cells across the chip area so that one or more target functions are optimized. Targets such as wire length, routability, critical path length and cell congestion have been considered in the past as optimization targets for the global placement step (a survey can be found in [106]).

The output from the global placement process does not necessarily satisfy the constraints, e.g., it might contain overlaps or unaligned cells. Therefore, at a second stage the output from global placement is *legalized*. Figure 4-1and Figure 4-2 depicts an example with the ibm05 benchmark circuit [83]. Figure 4-1 shows the global placement performed by NTUplace3 [33], while Figure 4-2 the output of legalization using the Tetris approach [73]. The usual performance metrics of algorithms in this category include the optimization targets of the global placement, most commonly *wire length* measured as follows: for each net in the circuit, the *half perimeter length* of the minimum bounding box totally enclosing the net is added. In order to have a performance metric that is oblivious to the particulars of the global placement, *cell displacement* is also used, measured as the *Manhattan distance* of the cell center before and after legalization. Intuitively, displacement declares that a legalization algorithm is efficient if it marginally disturbs the global placement. In this chapter we use both half perimeter wire length (*HPWL*) and displacement as performance metrics.

Although the circuit legalization problem forms the final step of the placement process, a number of circuit placers that work in an iterative manner such as NTUplace3 [33] and ePlace [103], incorporate a legalization step at the end of each iteration in order to estimate a “goodness” function for the global placement found so far. When used as an intermediate cost estimator, the time complexity of the

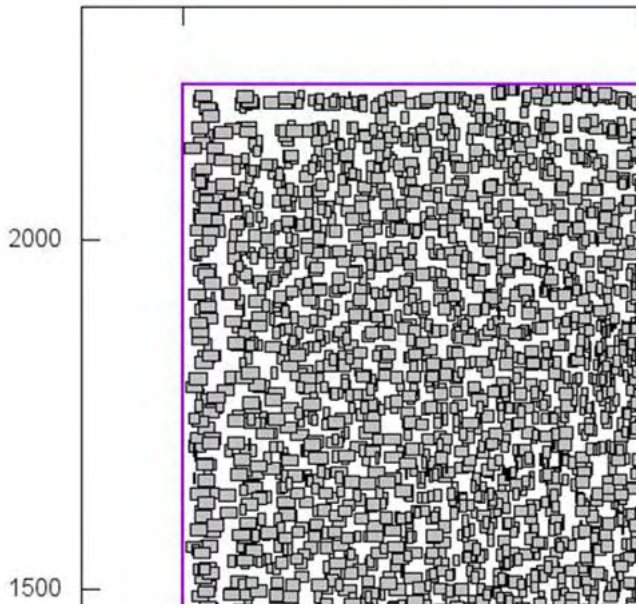


Figure 4-1 The output of NTUplace3 global placer

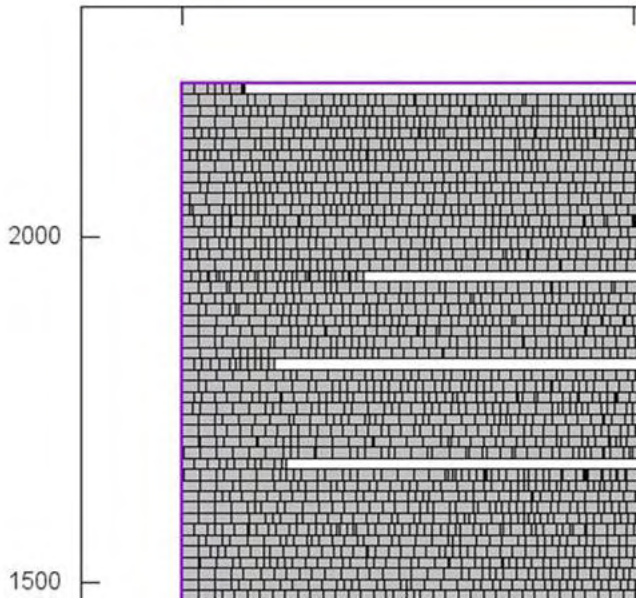


Figure 4-2 The output after legalization with Tetris algorithm

task a rather simple but fast legalization algorithm, e.g., Tetris [73], instead of a more complex option, e.g., Abacus [128] that is capable of achieving significantly better placement quality at the expense of execution time that is orders of magnitude higher.

Furthermore, in sub-90nm processes gate capacitances scale faster than interconnect capacitances, making the latter an ever-increasing component of the total switched capacitance, reaching over 20% in current process nodes. Since standard cell placement has a direct effect on the interconnect capacitance, it is becoming a power reduction vehicle targeting high activity factor nets such as nets on the clock distribution network [34], [121] and [145]. Weighing the capacitance of each net with the corresponding activity factor can lead to significant improvements as shown in [34]. Yet further research is warranted given the increased impact to total circuit power especially for interconnect dominated circuits such as networks-on-chip. As motivation and reference, the total switched capacitance and the total interconnect capacitance for a 45nm implementation of the ISCAS89 benchmark circuits are presented in Table 4-1. However, the total switched capacitance could not be obtained for the placement benchmarks (used by the placement community as reference) since they are not associated with an actual netlist.

Motivated by the aforementioned observation, in this chapter we focus on the legalization problem from the standpoint of providing sufficient solution quality at a small running time. In particular, we introduce heuristics that improve upon the solution quality of one of the fastest existing legalization algorithms (Tetris), while maintaining the distinct advantage of the algorithm in running time terms compared to the more complex methods. Our contributions include the following:

- We propose and evaluate 4 different heuristic adaptations of the basic Tetris algorithm, referred to as *Classic Tetris (CT)* algorithm. The heuristics are easy to implement and attack the problem of improving CT's performance from different angles.

- Aside from evaluating each heuristic as standalone we also evaluate their

combined effects. It turns out that the best performance is achievable through such combinations, while they also offer a variety of trade-offs between solution quality and execution time to choose from.

- All heuristics are evaluated over the 18 IBM benchmark circuits [83], with global placements obtained by Gordian [88] and NTUplace3 [33] algorithms. As there are no gate-level netlists for the aforementioned circuits, Activity Factors (AFs) of 1.0 have been assigned to nets that appeared to be parts of the clock network, while AFs of 0.1 have been assigned to the rest. Comparing the results against a state of the art legalization algorithm (Abacus [88]) and for NTUplace3 input, the performance achieved by one of the proposed heuristics (CC8-LR) was impressive. Specifically, whereas Abacus improvement over CT was 78.5% in HPWL, 97.5% in displacement and 86.7% in interconnect power, the heuristic achieved 75.1% in HPWL, 94.9% in displacement and 83.7% in interconnect power, but with a running time more than two orders of magnitude faster compared to Abacus. Similar (but to a lesser extent) large improvements over CT both in HPWL and displacement were also achievable with input from Gordian.

- We identified that certain heuristics were able to improve significantly the performance of CT in all four dimensions (HPWL, displacement, power, time), with the gains for the AC4-RR10% (Area Cut 4 – Restricted Row 10%) heuristic reaching (66%, 80%, 78%, 71%) with NTUplace3 input and (53%, 77%, 74%, 73%) with Gordian.

Although the CT legalizer is a rather old method, to the best of our knowledge this is the first work discussing heuristics to the basic algorithmic scheme that can significantly boost its performance quality wise while maintaining its fast running time. In [43] we presented a first limited report on the performance of the standalone heuristics. As it is shown in this chapter the combination of the heuristics is the one that leads to a drastic performance improvement capable of placing CT almost in par with the more sophisticated Abacus algorithm, particularly in the case of NTUplace3.

4.2. Tetris Heuristics

In this section we present the basic Tetris algorithm together with four proposed heuristics. We consider standard cell placement legalization whereby all cells have equal height and a solution consists of placing cells into rows of height equal to cell height.

4.2.1. Classic Tetris (CT)

The CT algorithm works as follows. First, it sorts all cells in increasing order of the x-coordinate of their center. Then assuming an initially empty chip area, it places the sorted cells consecutively starting from the one with the minimum x-coordinate. Once a cell is placed it is never again moved. In order to decide a single cell placement, CT calculates for each row the first available free position scanning the row from left to right, i.e., the first free position with the minimum x-coordinate. Some of these positions may be invalid because placing the cell there would result in exceeding the available chip area. Among the valid positions the one resulting in the minimum displacement, i.e., the one with the minimum Euclidean distance from the starting cell position, is selected.

We illustrate the process through an example that shows the CT legalization process over the initial placement depicted by Figure 4-3. The 6 cells involved in the example are named in order of their x-coordinates, therefore CT will first place cell A then cell B etc. Cell A will be placed at the beginning of the 4th row since its center lies within, thus, incurring the minimum displacement among all leftmost row positions. The process continues in a similar manner until the final placement is performed (Figure 4-5). Figure 4-4 illustrates the decision concerning cell D (cells E and F are omitted for clarity). The figure shows that once cells A, B and C are placed, among the 6 valid positions considered by the algorithm the one at the 4th row incurs the minimum displacement and is thus selected.

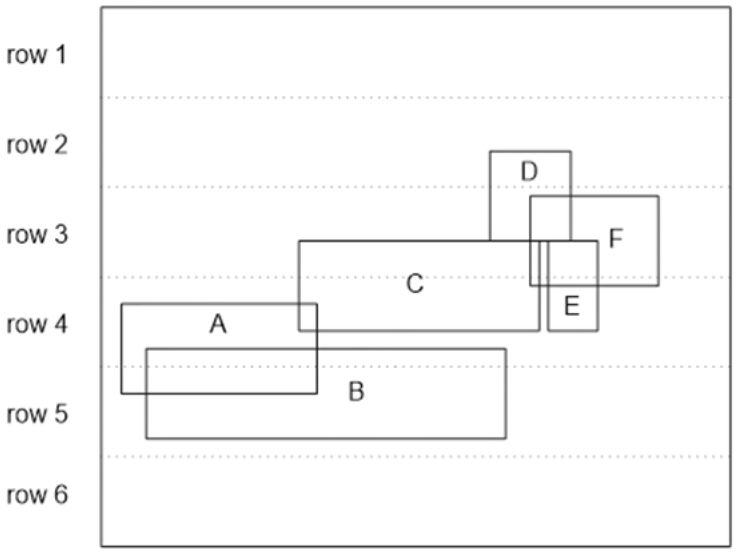


Figure 4-3 Initial placement

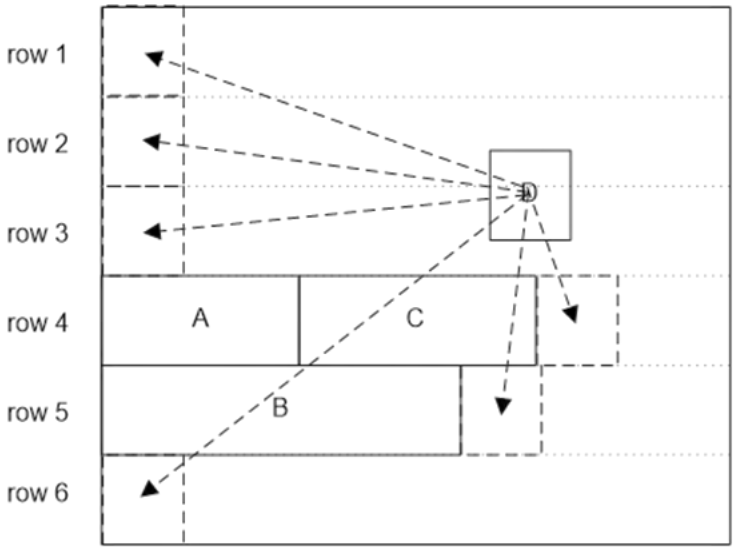


Figure 4-4 Possible positions for D cell

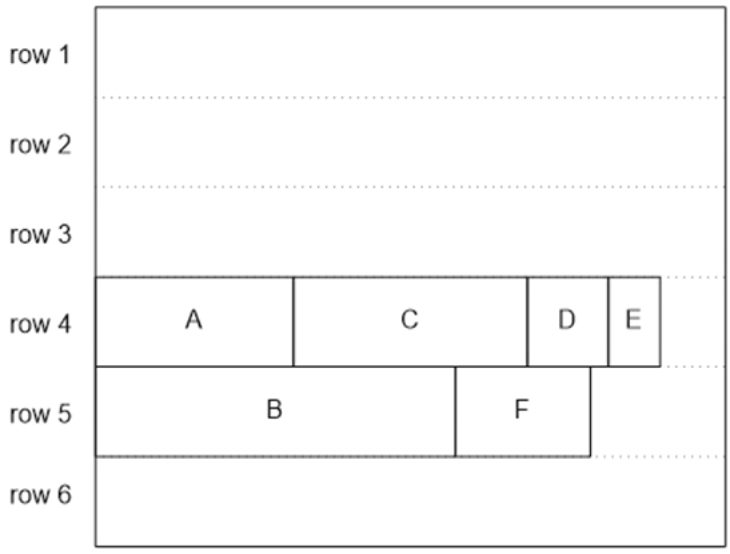


Figure 4-5 Final placement by CT

4.2.2. Restricted Row Heuristic (RR)

This approach works in the same manner as CT, with the exception being that rather than having all the rows as possible candidates, only *p*% of the rows are considered based on their proximity to the row where the cell's center initially resides. The rationale of the heuristic is two-fold. First, in CT (depending on the initial input) it is likely that a cell is placed in a row far away from the one it starts, due to the fact that the starting row as well as the ones close to it are relatively empty, while the distant one is more filled. Revisiting the example of Figure 4-4, cell D starts at the 3rd row but it is placed at the 4th row because this row is already filled up to a certain point while the 3rd one is empty. Even if CT decides on placement based on the smallest displacement, moving a cell far away from its original row might have adverse effects for the placement quality of subsequent cells. The second premise of the RR heuristic is orthogonal to the first one and aims at reducing the runtime of CT by limiting the number of rows the algorithm considers as placement candidates.

4.2.3. Left-Right Heuristic (LR)

The LR heuristic partitions the chip area in two. Cells with centers belonging to the left partition will be placed identically to CT, while the ones belonging to the right partition are placed in a similar to CT manner with the exception being that they will be sorted in decreasing (instead of increasing) order of x-coordinates and instead of placing them towards the leftmost legal positions, they will be placed towards the rightmost. We use the same example circuit of Figure 4-3 to illustrate the LR heuristic. Cells A and B belong to the left partition while the remaining to the right (Figure 4-6). In the final placement A and B will be placed as per CT, but the rest will be placed towards the right side of the chip area in reverse order, i.e., F first followed by E, D and C, resulting in the placement showed in Figure 4-7. The intuition behind the heuristic is that cells residing to the right partition probably have higher affinity with pins located at the right circuit edge. Therefore, moving them towards the right edge instead of the left one will result in smaller HPWL.

4.2.4. Area Cut-k Heuristic (ACk)

Under this approach the chip area is split into a $k \times k$ grid of equally sized rectangles. The algorithm then proceeds by performing CT for the local cells of each separate rectangle. Using the example circuit of Figure 4-3, Figure 4-8 shows the final placement produced by AC2. Notice that cells A and B belong to the bottom-left rectangle, C, E to the bottom-right and D, F to the top-right and are thus, placed in the aforementioned rectangles. Since cells might exceed rectangle boundaries (cell B in Figure 4-8) a strict order with which rectangles are considered is enforced. Namely, CT is applied starting from the top-left rectangle and continuing in a line by line manner. In the extreme case where a cell can't be placed within its rectangle because the whole rectangle area is filled, all circuit rows are considered as placement candidates. The premise of this heuristic is that by vertically splitting the chip area a bound on the displacement over the x-axis is essentially introduced. The same holds true for the y-axis displacement and the horizontal partitioning, an idea that is also captured by RR heuristic.

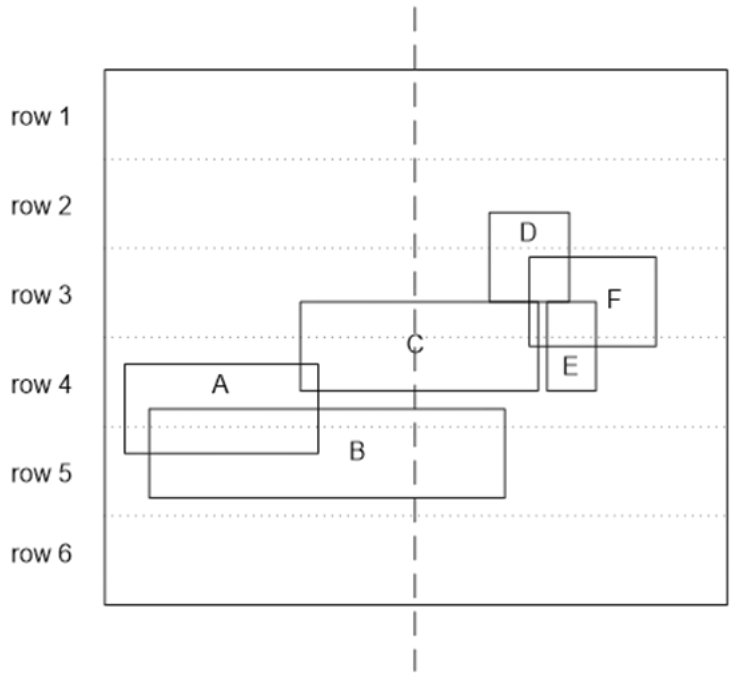


Figure 4-6 The initial placement and the virtual split of the chip area in two equal partitions.

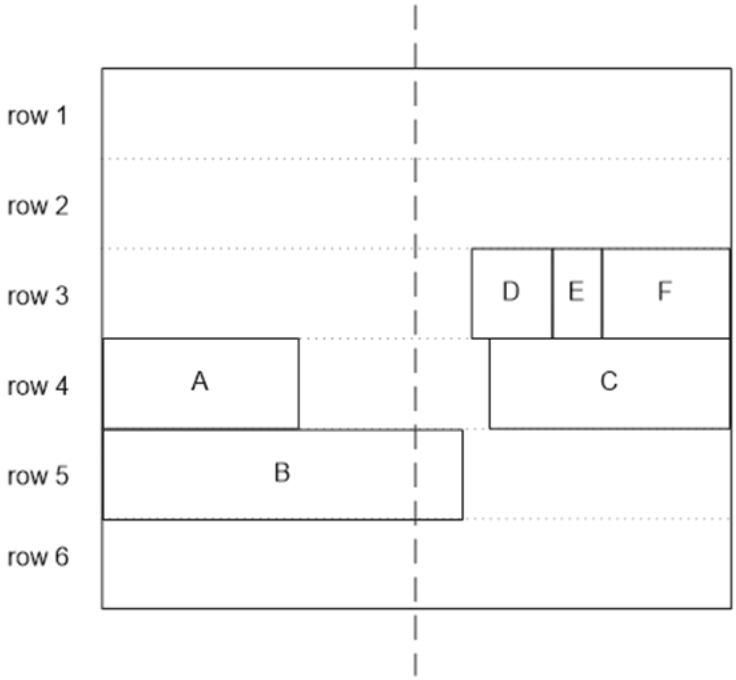


Figure 4-7 Final placement by LR heuristic

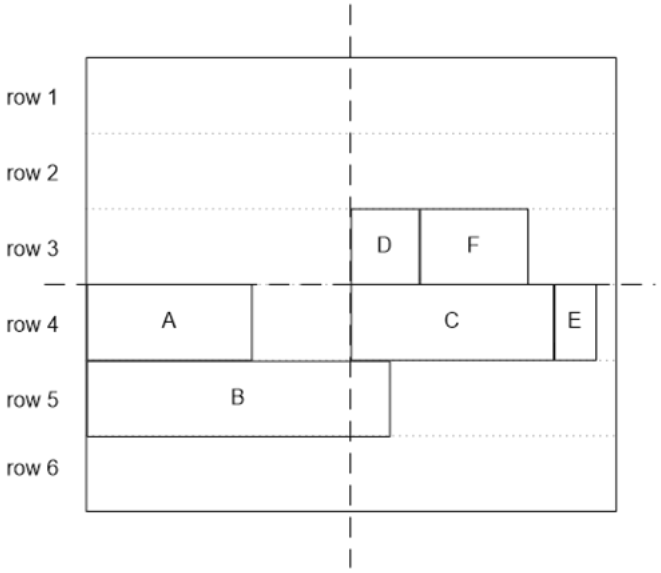


Figure 4-8 The split in 2×2 grid and the output of AC2 heuristic.

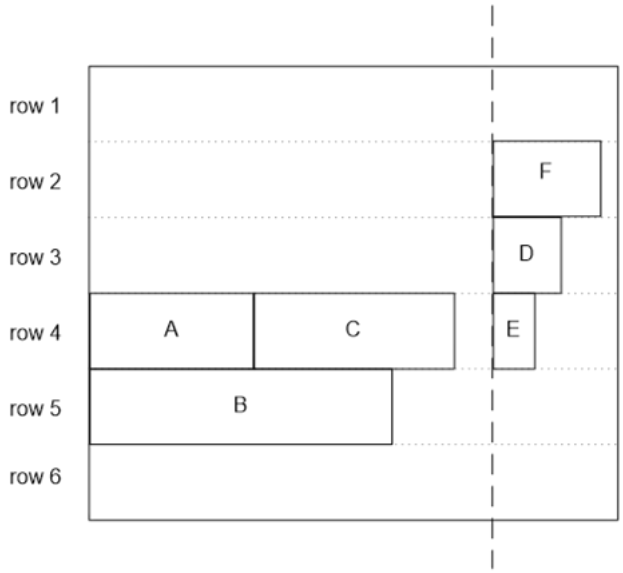


Figure 4-9 The vertical split in two areas and
the placement performed by CC2

4.2.5. Cell Cut-k Heuristic (CCk)

This heuristic splits the sorted by x-coordinate cell list into k equally populated lists (let L_i, $1 \leq i \leq k$). For each L_i it calculates the aggregated area of its cells (let $area(L_i)$). It then splits the chip area vertically into k partitions (let R_i, $1 \leq i \leq k$), such that the area of R_i is given by (4.1):

$$area(R_i) = (area(L_i) \times area(chip)) / \sum_{x=1}^{k} area(L_x) \quad (4.1)$$

The heuristic proceeds by applying CT in order to place the cells belonging to L_i at R_i. Figure 4-9 illustrates for the example circuit of Figure 4-3 the final placement produced by CC2. Notice, the vertical partitioning into two unequal areas that follows the size proportion of cells A, B and C versus the size of cells D, E and F. The intuition behind this heuristic is that by defining vertical splits in a proportional manner, partitions will have equal density (as opposed to ACk heuristic), thus, leading to better results.

4.3. Experiment Results

We evaluated the performance of the heuristics using the ISPD04 dataset and in particular the ibm01 – ibm18 circuits [83]. Global placements were produced by running Gordian and NTUplace3. Three performance criteria were used, namely: total cell displacement (Displace) measured using Manhattan distance, total net half perimeter wire length (HPWL) and running time (Time). For each circuit we recorded the percentage of performance improvement of each heuristic (H) over CT as follows:

improvement = (*performance*(*CT*) – *performance*(*H*)) / *performance*(*CT*).

In order to characterize the total performance of the heuristics over the entire dataset we used the average improvement experienced over all circuits.

4.3.1. Evaluating heuristics combinations

Motivated by the results of Table 4-2 we proceeded by evaluating heuristic combinations. Namely, for each of the partitioning heuristics (ACk and CCk) we tested

the performance when combined with LR and RR. The rationale was the following: since LR, RR and the partitioning heuristics all managed to improve HPWL and displacement when used in standalone mode and they attack the problem from different angles, it is likely that their combination will further improve performance on these two metrics. Furthermore, for the case of RR combinations it is likely that it will further improve time performance of the partitioning heuristics. Based on Table 4-2 we only considered the application of RR10% which was shown to be the most promising RR alternative.

We tested two-part combinations whereby for instance AC2-LR and AC2-RR10% denote that the chip area is split according to AC2 and for each resulting rectangle legalization is performed using LR and RR10% respectively (similarly for CCk heuristic). We also included in the experiments three part combinations whereby for instance AC2-LR-RR10% means that the chip area is partitioned according to AC2 and at each rectangle, LR was used for legalization but with the constraint implied by RR10%. Table 4-3 summarizes the results for ACk based combinations, while Table 4-4 for CCk based. In both tables LR and RR10% were also included for comparison reasons. Boldfaced values indicate dominating heuristics on each table separately.

As observed in Table 4-3 and IV, applying both LR and RR10% simultaneously over ACk and CCk heuristics gave results that were constantly dominated by the ones produced when applying RR10% only. Thus, the three-part heuristics don't account for valid alternatives. Concerning ACk alternatives, the ones based on AC4 seem to offer the most viable solutions. Notice, that AC3-RR10% belongs in the dominating set due to a marginally better performance by 0.19% against AC4-RR10% in the Gordian case. Furthermore, the application of LR and RR10% works as per our motivation. Namely, comparing AC4-RR10% against AC4 improved performance is achieved across all three metrics, while the time gains are particularly impressive with an extra of roughly 20% in NTUplace3 and 26% in Gordian. AC4-LR achieves the best performance HPWL and displacement wise with an extra of at least 10% compared to standalone AC4 in all cases.

Similar observations hold for CCk based combinations in Table 4-4. The best performance in HPWL and displacement is obtained by applying LR over CC8, but at a high time performance degradation compared to CT. Applying RR10% drastically cuts down time cost. For instance whereas CC8 accounts for a time increase compared to CT of 67.58% and 53.77% with Gordian and NTUplace3 input respectively, CC8-RR10% accounts for 45.53% and 42.42% time reduction.

In order to clarify which heuristics account for the best alternatives, in Table 4-5 we record the heuristics that produced dominating results over Table 4-2, Table 4-3 and Table 4-4 cumulatively. We also recorded the performance of the Abacus legalization method for comparison reasons. Boldfaced are heuristic results that provide the most interesting trade-offs. CC4-LR is excluded from this list because it offers only marginal improvement of less than 1% in HPWL and displacement terms compared to AC4-LR while its time degradation is roughly 100% compared to the latter. AC3-RR10% is also excluded since compared to AC4-RR10% it only provides an extra 0.19% improvement in time for the Gordian case while losing in all other comparisons. Lastly, CC8-RR10% is excluded because compared to AC4-LR it is slightly better time wise, but worse in HPWL and displacement.

From Table 4-5 it is evident that the best performance on HPWL and displacement is offered by the Abacus algorithm, however, its time cost is larger by two orders of magnitude compared to CT. For the best performing heuristics and NTUplace3, interconnect power is presented in Table 4-6 as a fraction of the one achieved by CT. As far as the heuristics of this chapter are concerned, CC8-LR and AC4-RR10% offer different trade-offs between HPWL/displacement and time. CC8-LR has the best performance in HPWL and displacement terms but incurs an extra running cost, whereas AC4-RR10% offers the best time improvement while also improving (to a lesser extend) HPWL and displacement. AC4-LR offers a trade-off in between CC8-LR and AC4-RR10%. Summarizing the results we can state the following:

- Compared to CT, CC8-LR, AC4-LR and AC4-RR10% all offer significantly better performance in terms of placement quality (HPWL and displacement) while the AC variants greatly outperform CT also in running time;

- Compared to Abacus, CC8-LR offers the best Tetris heuristic alternative. It is noteworthy that in the case of NTUplace3 the performance of CC8-LR in HPWL and displacement is very close (less than 4%), while its running time is still faster by two orders of magnitude;

- Lastly, all the proposed heuristics are simple in nature and can be easily incorporated in current placement suits.

Circuit	Interconnect Capacitance	Diffusion Parasitics	Pin Parasitics	Load Capacitance	Interconnect Contribution
s27	7.82	0.21	0.08	40.59	15.6%
s298	114.56	1.00	1.20	476.74	19.0%
s344	117.34	1.00	2.13	487.69	18.9%
s349	113.86	1.48	2.09	450.31	19.5%
s382	159.30	1.13	3.36	622.27	19.8%
s400	157.15	0.97	1.55	633.44	19.6%
s420	165.20	1.74	2.17	612.37	20.7%
s526	213.89	1.60	2.15	831.85	20.1%
s641	255.99	6.67	5.49	1032.50	18.9%
s713	275.59	6.99	5.74	1079.60	19.4%
s820	415.13	3.29	3.73	1339.54	23.3%
s953	443.86	3.10	4.09	1277.94	25.4%
s1196	656.49	3.62	3.92	1546.06	29.5%
s1238	670.32	3.66	3.71	1587.70	29.4%
s1423	479.44	4.59	3.03	1600.04	22.7%
s1488	843.36	6.14	5.72	2038.09	28.9%
s5378	2338.24	15.76	16.33	7142.01	24.3%
s9234	1799.21	24.02	15.53	7471.00	19.0%
s13207	5662.73	50.13	44.86	17455.81	24.1%
s15850	6795.52	75.66	54.23	20618.92	24.3%
s35932	14742.85	101.24	144.25	41960.02	25.6%
s38417	15849.30	130.49	117.95	51730.57	23.1%
Average					**22.3%**

Table 4-1 Contribution of interconnect capacitance to total switched capacitance (in pF)

Heuristics	Gordian			NTUplace3		
	HPWL	Displace	Time	HPWL	Displace	Time
LR	35.54%	49.10%	-9.04%	46.14%	48.09%	-11.41%
RR10%	**20.26%**	**12.22%**	**61.43%**	**30.36%**	**11.62%**	**40.30%**
RR20%	13.22%	7.66%	44.85%	18.60%	7.21%	24.80%
RR30%	7.85%	4.33%	29.03%	10.89%	4.02%	3.13%
AC2	36.51%	52.77%	12.15%	45.42%	56.30%	21.28%
AC3	47.75%	69.74%	34.78%	57.19%	71.89%	43.62%
AC4	**49.65%**	**74.19%**	**47.23%**	**61.84%**	**78.28%**	**51.13%**
CC2	35.34%	51.94%	-62.30%	43.52%	55.74%	-62.30%
CC4	**50.38%**	**74.55%**	**-63.66%**	**60.40%**	**77.90%**	**-51.58%**
CC8	**57.11%**	**85.69%**	**-67.58%**	**68.66%**	**88.55%**	**-53.77%**

Table 4-2 Performance improvement of standalone heuristics over CT

Heuristics	Gordian			NTUplace3		
	HPWL	Displace	Time	HPWL	Displace	Time
LR	35.54%	49.10%	-9.04%	46.14%	48.09%	-11.41%
RR10%	20.26%	12.22%	61.43%	30.36%	11.62%	40.30%
AC2	36.51%	52.77%	12.15%	45.42%	56.30%	21.28%
AC2-RR10%	44.58%	58.41%	68.07%	56.84%	61.52%	67.72%
AC2-LR	56.04%	77.38%	1.29%	67.24%	80.45%	-8.29%
AC2-LR-RR10%	22.51%	46.96%	62.58%	24.70%	48.86%	63.25%
AC3	47.75%	69.74%	34.78%	57.19%	71.89%	43.62%
AC3-RR10%	**51.76%**	**73.00%**	**74.14%**	**63.75%**	**74.99%**	**69.27%**
AC3-LR	59.66%	85.82%	23.63%	71.49%	87.78%	19.22%
AC3-LR-RR10%	37.03%	62.80%	67.28%	41.75%	65.06%	68.55%
AC4	49.65%	74.19%	47.23%	61.84%	78.28%	51.13%
AC4-RR10%	**53.55%**	**77.29%**	**73.95%**	**66.54%**	**80.58%**	**71.38%**
AC4-LR	**59.87%**	**87.40%**	**43.99%**	**72.81%**	**90.66%**	**34.55%**
AC4-LR-RR10%	43.64%	70.70%	72.39%	50.59%	73.07%	67.01%

Table 4-3 Performance improvement of AC heuristic combinations over CT

Heuristics	Gordian			NTUplace3		
	HPWL	Displace	Time	HPWL	Displace	Time
LR	35.54%	49.10%	-9.04%	46.14%	48.09%	-11.41%
RR10%	**20.26%**	**12.22%**	**61.43%**	**30.36%**	**11.62%**	**40.30%**
CC2	35.34%	51.94%	-62.30%	43.52%	55.74%	-62.30%
CC2-RR10%	**41.96%**	**56.10%**	**48.53%**	**52.75%**	**59.75%**	**46.73%**
CC2-LR	55.81%	77.11%	-60.59%	67.23%	80.56%	-59.64%
CC2-LR-RR10%	18.92%	44.42%	47.58%	19.07%	46.31%	44.26%
CC4	50.38%	74.55%	-63.66%	60.40%	77.90%	-51.58%
CC4-RR10%	**51.55%**	**75.31%**	**51.82%**	**62.38%**	**78.69%**	**45.89%**
CC4-LR	**60.41%**	**87.58%**	**-60.93%**	**72.84%**	**90.74%**	**-60.68%**
CC4-LR-RR10%	41.53%	68.76%	49.88%	45.87%	70.84%	42.82%
CC8	57.11%	85.69%	-67.58%	68.66%	88.55%	-53.77%
CC8-RR10%	**57.15%**	**85.72%**	**45.53%**	**68.73%**	**88.56%**	**42.42%**
CC8-LR	**61.87%**	**91.85%**	**-72.18%**	**75.15%**	**94.89%**	**-63.20%**
CC8-LR-RR10%	53.10%	82.01%	42.97%	61.64%	84.28%	38.75%

Table 4-4 Performance improvement of CC heuristic combinations over CT

Heuristics	Gordian			NTUplace3		
	HPWL	Displace	Time	HPWL	Displace	Time
Abacus	**83.06%**	**98.08%**	**-2587.94%**	**78.82%**	**97.53%**	**-5481.04%**
AC3-RR10%	51.76%	73.00%	74.14%	63.75%	74.99%	69.27%
AC4-RR10%	**53.55%**	**77.29%**	**73.95%**	**66.54%**	**80.58%**	**71.38%**
AC4-LR	**59.87%**	**87.40%**	**43.99%**	**72.81%**	**90.66%**	**34.55%**
CC4-LR	60.41%	87.58%	-60.93%	72.84%	90.74%	-60.68%
CC8-RR10%	57.15%	85.72%	45.53%	68.73%	88.56%	42.42%
CC8-LR	**61.87%**	**91.85%**	**-72.18%**	**75.15%**	**94.89%**	**-63.20%**

Table 4-5 Dominating heuristics

Circuit	Abacus	AC4-RR10%	CC8-LR
ibm01	18%	28%	21%
ibm02	15%	23%	17%
ibm03	21%	31%	24%
ibm04	19%	27%	22%
ibm05	15%	20%	17%
ibm06	13%	20%	15%
ibm07	13%	24%	18%
ibm08	15%	24%	17%
ibm09	17%	29%	21%
ibm10	11%	18%	13%
ibm11	11%	19%	13%
ibm12	14%	23%	17%
ibm13	17%	30%	20%
ibm14	8%	16%	12%
ibm15	8%	16%	10%
ibm16	9%	18%	11%
ibm17	10%	19%	16%
ibm18	8%	17%	10%
Average	**13.3%**	**22.2%**	**16.3%**

Table 4-6 Interconnect power of heuristics as a fraction of CT interconnect power

5. Parallelizing Legalization Procedure

5.1. Motivation

In standard cell placement, cells are assumed to be of equal height (different width) and the chip area is split into rows of height equaling cell height. The target of placement is to arrange all the cells of the circuit within the chip area so that they are aligned to rows, no overlaps exist and some target function is optimized. Candidates for optimization include wire length, cell congestion, routability, critical path length etc. [106] provides a comprehensive survey on cell placement.

The final placement of a circuit is most commonly achieved in stages. The first stage (global placement) produces an arrangement of cells on the chip plane so that cells are sufficiently spread from each other and the optimization criteria are met. This output, is seldom a valid placement and usually contains overlaps and misaligned cells. For this reason the final stage in the placement process is legalization whereby the target is to restore validity constraints while reaching a final placement that is as close as possible to the one produced by the global placer.

Legalization algorithms can also be found as components of iterative global placers, whereby they are applied in order to estimate the "goodness" of intermediate solutions. Especially in this case, the trade-off between solution quality and running time is particularly important, since the legalization scheme isn't meant to produce a final solution once, but will be called repeatedly to get an estimation of the actual performance achieved by each iteration. Different approaches to deal with this trade-off can be found in the literature. On one extreme ePlace [103] adopts a variant of a simple greedy method called Tetris [73] that is known to be very fast but produce low quality solutions. On the other hand global placers such as Kraftwerk2 [129] use the more sophisticated Abacus [128] method that is known to achieve top quality solutions but at runtime that is roughly two orders of magnitude compared to Tetris.

In this chapter we turn our focus on speeding up the legalization algorithms that offer top quality solutions, such as Abacus, using parallelization. Our approach is based on

splitting the chip area vertically into subareas to be considered independently. Clearly, the algorithmic details might preclude the existence of purely independent subareas. In this case any conflicts that exist are resolved in a sequential manner. We demonstrate the efficacy of our parallelization approach by implementing it over the Abacus algorithm. For ease of reference we term the resulting method Domocus. Our contributions include the following:

- We propose a lock-free parallelization framework that can be applied over various legalization schemes.

- We implemented and tested the framework over a legalization algorithm (Abacus) known for its solution quality but also slow running time.

Results indicate that Domocus achieves good scalability, while retaining at large the solution quality of the Abacus scheme, even improving it from certain performance aspects.

5.2. ABACUS overview

Here we provide a brief overview of Abacus' functionality through the illustrative example of Figure 5-1. A comprehensive description can be found in [128]. The Abacus algorithm works as follows. First all cells are sorted according to their x-axis coordinates. [128] indicates that the algorithm should be run both by sorting in increasing and in decreasing order (choosing the best among the two). In this chapter we only consider for simplicity increasing order. The algorithm then proceeds by placing cells one by one. Each cell placement is performed as follows. The algorithm starts with the row where the cell center currently belongs and attempts to place it there calculating an initial displacement cost *C*. It then examines the cost of placing the cell in the rows lying above the starting one and then the ones that rest downwards. If at any point when considering a direction (upwards or downwards) the row examined incurs a higher cost than the minimum one calculated already, the algorithm stops examining any more rows on this direction. The aim of this early termination criterion is to reduce the running time of the algorithm.

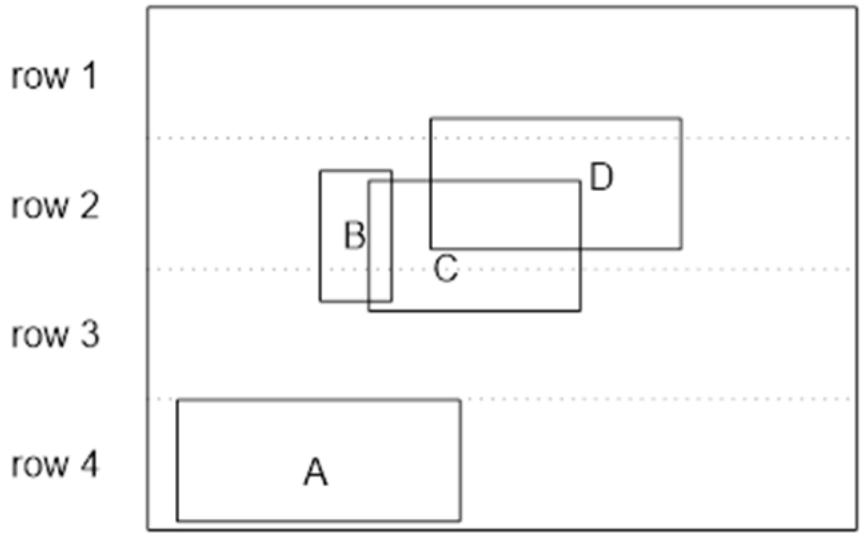


Figure 5-1 Initial global placement.

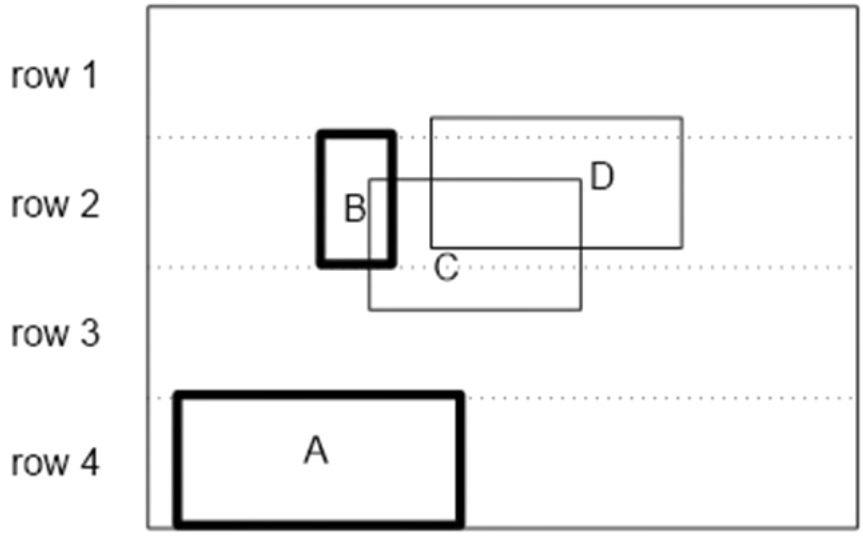


Figure 5-2 Placement of A and B cells.

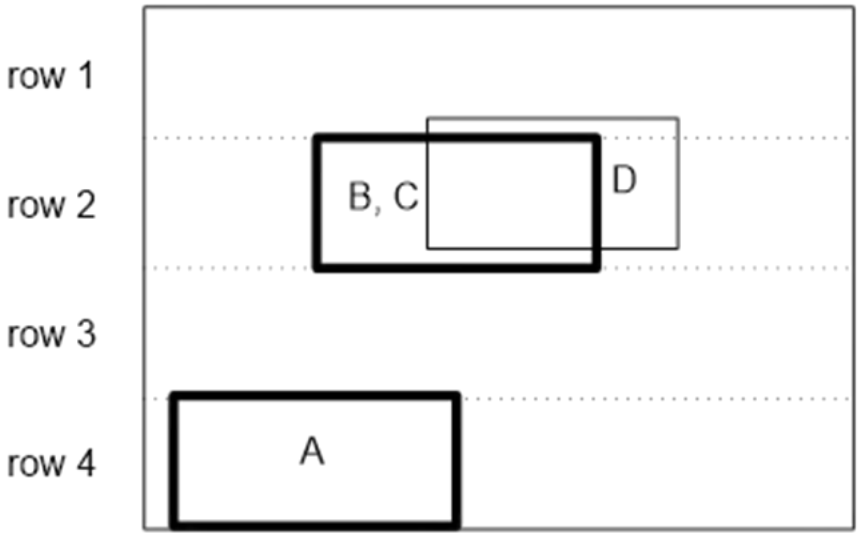


Figure 5-3 Cluster formation with B and C cells

Calculating row displacement cost involves finding the best cell position within the examined row. The first step towards this is to consider the ideal placement which is the one obtained by moving the cell vertically so that it is row aligned. In case this position incurs no overlaps with previously placed cells the position is retained. Otherwise, a cluster is formed with the overlapping cells (or clusters) and then the best cluster position is calculated within the specified row. Cells within a cluster maintain their x-coordinate order.

Considering the example global placement of Figure 5-1, whereby the four cells are named in order of increasing x-coordinates, cell A will be placed first, followed by cell B, resulting in the intermediate instance shown by Figure 5-2. When cell C will be considered for placement at row 2 it is evident that its ideal position at row 2 overlaps with cell B. For this reason C is in a sense appended at the end of cell B and the two cells form a cluster Figure 5-3. Continuing the example, when cell D will be considered for placement at row 2, it will be added to the (B, C) cluster as shown in Figure 5-4. As a

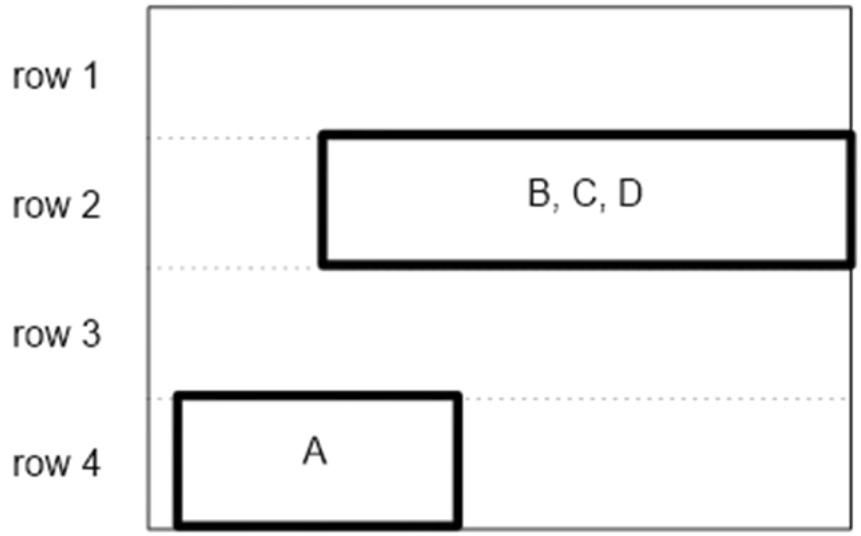


Figure 5-4 D is appended to the cluster

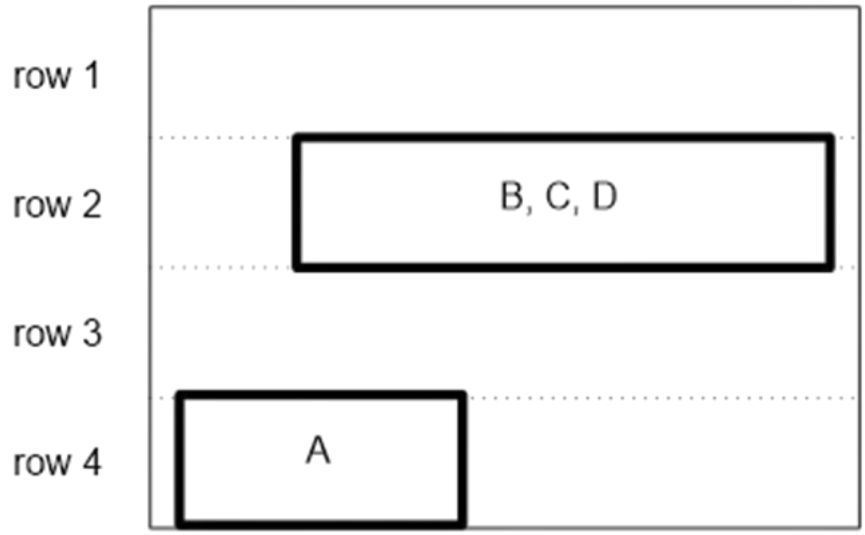


Figure 5-5 Cluster moved to optimal position.

final step, the whole cluster (B, C, D) will be moved towards the position that minimizes the cumulative distance of all cluster cells from their ideal positions in the row.

5.3. DOMOCUS parallel algorithm

Parallelizing Abacus poses interesting dilemmas. Recall that the algorithm considers cells in a sorted manner and for each cell, potentially all candidate rows are checked, using an early termination criterion. It is apparent that two possible parallelization granularities can be used. The first is at the group of cells level, whereby each thread will be tasked with the placement of different cells (coarse grained parallelism) and the other one is at a cell level, whereby the threads will be tasked to cumulatively calculate a single cell's placement (fine grained parallelism). Although the fine grained approach requires no particular synchronization overhead (one can assign different threads for different row cost calculations), it doesn't scale well, due to the early termination criterion. Consider for instance the case where the best displacement is given by forming a cluster in the row where the cell currently resides. In this case three rows will be totally checked (the current row, the one above it and the one below it), which means that in systems with more than three CPU cores, the remaining cores will stay idle. For this reason we chose to follow the coarse grained approach.

Using coarse grained parallelism in a straightforward manner introduces significant synchronization overhead, since it must be ensured that no two threads will place their cells in overlapping positions or incorporate them at the same cluster. Instead, we decided to follow an alternative lock-free approach. Specifically, we partition the chip area into N equally sized vertical zones, with N equaling the number of available CPU cores. Each zone is processed by a separate thread by applying the Abacus algorithm over the cells contained in the zone sorted by x-coordinate.

When considering a specific cell-row placement on the zone it belongs to, two cases rise: (i) the cell is placed in the row so that neither the cell, nor the cluster it was added to (if it did) exceeds zone boundaries; (ii) the placement results in the cell or a cluster exceeding zone boundaries. In case of (i) the placement is considered valid, otherwise if (ii) holds, the placement is invalid and discarded. In this manner it is possible that for some cells no eligible positions are found. Once all threads finish with their assigned placements, any cells that remain unplaced are handled in a second sequential step without zone boundary restrictions.

With the aforementioned methodology zones (and the placements within them) are completely independent thus, no synchronization is required and the scheme is completely lock free. It is worth noting that the previously described mechanism that consists of zone partitioning and a second sequential step to deal with unplaced cells can be applied to other legalization schemes (aside from Abacus) provided the schemes themselves don't follow their own chip area partitioning. For instance another straightforward candidate is [73].

In this chapter we focused on speeding up the execution of Abacus legalization. The resulting parallel algorithm Domocus, might divert in performance from the sequential Abacus due to two reasons. First, candidate rows deemed ineligible due to boundary restrictions, might have been used by the sequential scheme. Secondly, the existence of unplaced cells that are handled at the end, changes the order with which Abacus considers cells for placement. In the following section we evaluate any possible negative effects on solution quality due to parallelization, but also the achievable speed-up.

5.4. Experiment

We implemented Domocus and compared its performance to sequential Abacus across the following three metrics: displacement, half perimeter wire length (HPWL) and time. Displacement was measured using the Manhattan distance between the initial cell position as defined by the global placer and the resulting one by the legalization process. HPWL is a commonly used metric in the literature in order to estimate the total required wire length. Namely, for each net in the circuit the minimum bounding rectangle that encloses all its cells is calculated and its half perimeter is accumulated. We used the 18 ibm circuits as benchmarks [83], assuming an initial global placement obtained by the algorithm in [88]. Experiments were run on a Linux server with two 6-core Intel Xeon E5-2630 CPUs running at 2.3GHz using hyper threading (12 physical cores total). Parallelization was done using OpenMP.

Figure 5-6 presents the performance in HPWL and displacement terms of Domocus (for 1, 2, 4, 8 and 12 threads) as a percentage improvement over Abacus, using the following formula:

100(*perf*(Abacus) - *perf*(Domocus)) / *perf*(Abacus)

The figure shows the averaged improvement results over all 18 circuits. First, it is clear that any performance difference is rather small which is particularly encouraging for our method. To explain the results we should notice that as the number of zones (threads) increases, so will the number of unplaced cells. These cells are placed in a sequential out of order manner. This out of order placement seems to have a small negative effect on HPWL while in displacement terms it doesn't have a great impact. In fact the Domocus accounts for a small improvement in displacement. This is presumably due to the fact that by enforcing zone boundaries, a constraint is effectively imposed on the maximum displacement along the x-axis.

Figure 5-7 plots the speedup of Domocus over sequential Abacus. Observe that with 2 and 4 threads super-linear speedup is achievable, highlighting the merits of our approach. With 8 and 12 threads the performance is not similarly outstanding since the size of the second sequential part increases (more cells are left unplaced as previously explained). However, this observation doesn't diminish the value of our approach which is solidly built upon the super-linear speedups experienced with 2 and 4 threads. Rather, it provides a direction for future work whereby the second sequential step will be also parallelized in a similar lock-free manner.

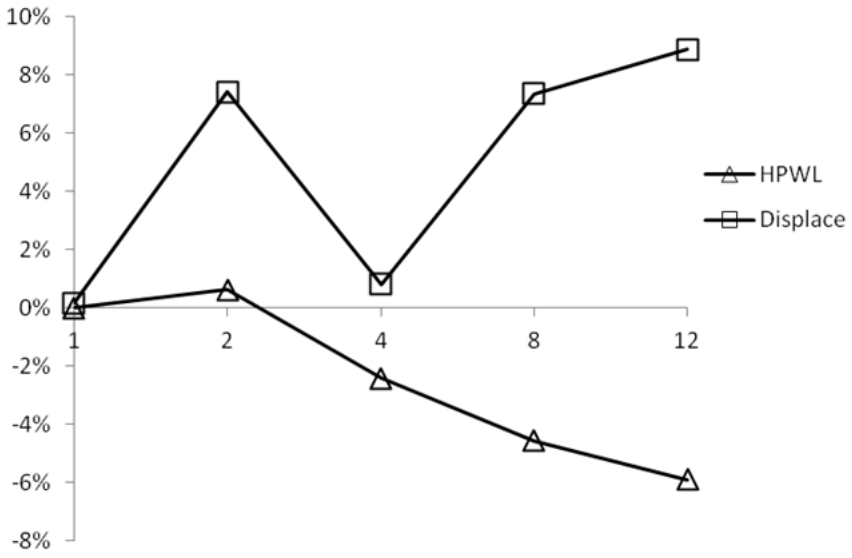


Figure 5-6 Average performance improvement (%) for the HPWL and displacement metrics (x-axis shows thread number).

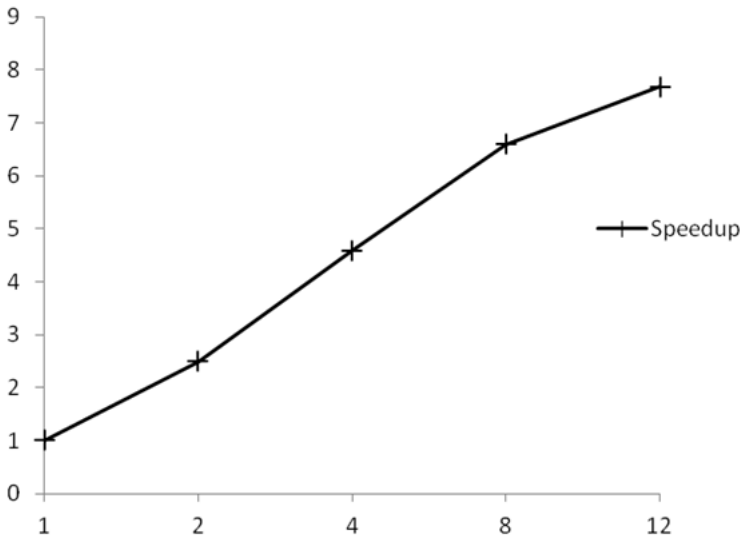


Figure 5-7 Speedup over the sequential execution (x-axis shows thread number).

5.5. IC Placement over the Cloud

Domocus performance was also tested using Google Cloud Engine (GCE) [68]. GCE provides high-performance virtual machines with predefined configurations. Experiments were run on a Linux standard machine type with 8 virtual CPUs, 30 GB of memory and 20GB SSD persistent disk (n1-standard-8). The virtual CPU is implemented as a single hardware hyper-thread on a 2.6 GHz Intel Xeon E5 (Sandy Bridge). Europe-west1-b zone was selected to host our experiments. Choosing a region which is close to our point of service might decrease network latency. Figure 5-8 shows the algorithmic performance regarding running time using ISCAS89 benchmark circuits (for 1, 2, 4 and 8 threads). The first machine (mcA) was described in 5.4 while mcB is the aforementioned one. Despite that there are no significant differences between the two instances it is obvious that GCE is a good alternative. Beside execution time, GCE gives users the possibility to rapidly build and test legalizers. In most cases Cloud's expenses can be a block for users, however, bandwidth, database storage and disc space is not IC placement concern. Aside of typical operation costs like benchmark downloading and essential application installation. Figure 5-9 presents the overall testing cost at n1-standards-8 instance. Results point out that Cloud environment fits the requirements of IC placement with faint expenses.

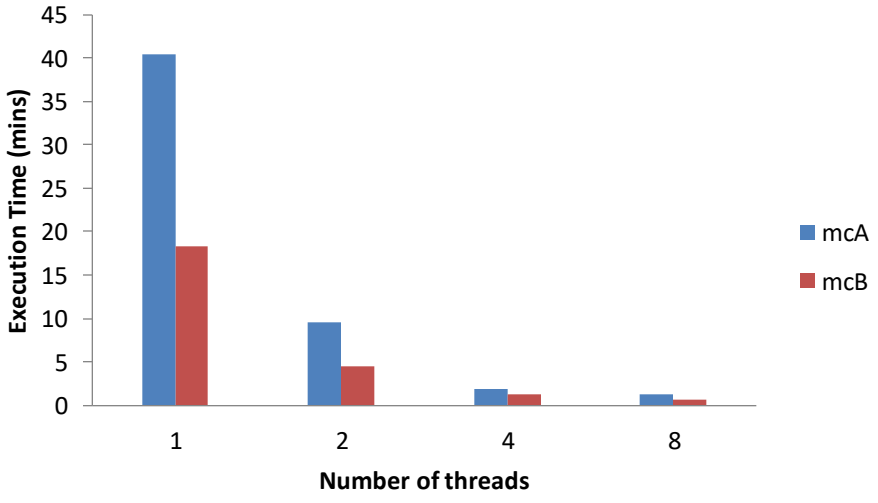


Figure 5-8 Cloud vs. Dedicated instance execution time

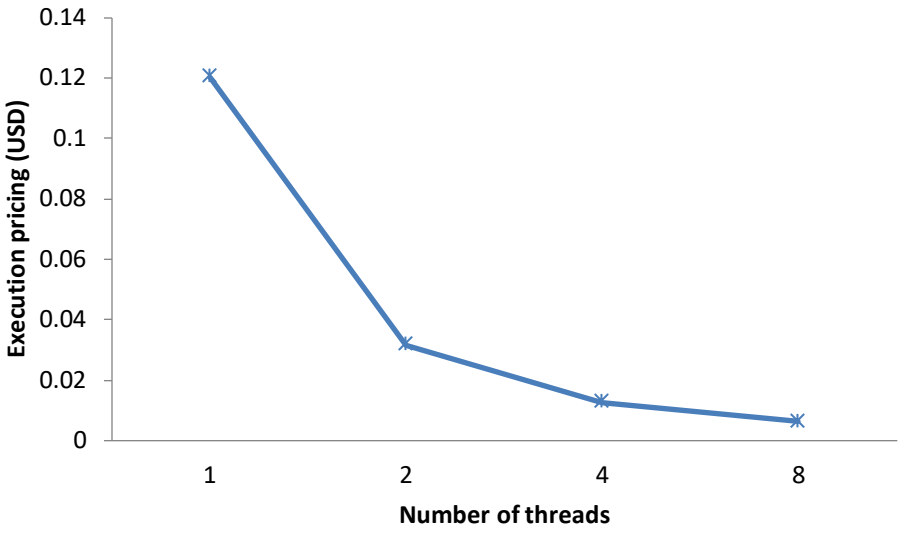


Figure 5-9 Cloud expenses testing Domocus legalizer

6. Job Scheduling over the Cloud

6.1. Overview

The main goal of a job scheduling algorithm is to provide a high computation performance in parallel with optimal system efficiency and profit. Traditional scheduling algorithms don't meet Cloud environment requirements since users have to pay for resources used, based upon time and due to high communication cost. In the Cloud, clients may use thousands of different type of resources, as a result job scheduling is quite challenging affecting servers' utilization, energy efficiency, load balancing of data centers and QoS which is determined by the user which usually includes execution time. Job scheduling in Cloud environment is primarily achieved via batch mode heuristic scheduling algorithms (BMHA) [155].

In BMHA jobs are grouped, queued and combined when they arrive to the system. These types of algorithms start after a fixed time period and constitute variations of well-known scheduling algorithms like First Come First Server algorithm (FCFS) and Round Robin (RR). In [4] authors show that a generalized priority scheduling algorithm which considers execution cost and execution time is more efficient than FCFS and RR. The min-min algorithm is extremely simple and is used as a basis for many others. It works fast and chooses small tasks to be executed first. The algorithm starts with a list of tasks and a list of virtual machines (VMs). Afterwards, finds the VM in which all tasks have the smallest completion time and assign to this VM the smallest task. Max-min algorithm works in the same manner as min-min except that it chooses large task to be executed first. Heterogeneous Earliest Finish Time (HEFT) [142] differs from the aforementioned two algorithms since it use an Directed Acyclic Graph (DAG) according to which each task is given an priority which is used for the VM selection and submission. SHEFT [101] is an expansion of [142] where compute resources are unbound. Cloud environment is modelled by partitioning all resources into a number of clusters. Reliable Scheduling Distributed in Cloud Computing [50] is used for both load balancing and processing time reduction by dividing the major jobs into sub jobs. Scheduling of each job is achieved by calculating the request and acknowledge time in the form of a shared job. In [43] a priority and admission control based service scheduling policy is proposed

which targets QoS satisfaction and systems throughput maximization. Scheduling group of tasks is achieved in [126] where two objectives are taking into account, resource cost and computation performance. Multi-objective optimization used in [151], [132] and [137]. [151] use map reduce to improve energy efficiency in data centers in conjunction with system performance. Using two interdependent strategies [132] is a cost-efficient scheduling algorithm in Cloud environment. At first, it maps tasks to most cost-efficient VMs and continues by reducing the operation cost of non-critical tasks. Makespan and energy consumption constitute the optimization objectives in [137] where a hybrid Genetic algorithm is used to provide pareto solutions. Finally, probabilistic-nature inspired techniques are used in [4], [139] and [97] where Particle Swarm, Ant Colony and Artificial Bee Colony optimization are applied respectively.

6.2. Overview on Scheduling Heuristics for Live Video Transcoding on Cloud Edges

Efficient video delivery involves the transcoding of the original sequence into various resolutions, bitrates and standards, in order to match viewers' capabilities. Since video coding and transcoding are computationally demanding, performing a portion of these tasks at the network edges promises to decrease both the workload and network traffic towards the data centers of media providers. The number of transcoding tasks hosted by edges is dictated by their processing requirements concerning speeding up of video coding and transcoding. An avid research exists on parallelizing video coding with approaches varying from coarse grained parallelism, whereby parallelism is considered at the level of group of Macroblocks (H.264/AVC) or Coding Tree Units (CTUs in HEVC), to finer grained parallel approaches implementable within a block of pels. Examples of coarse grained parallelization include slices, tiles and wavefront in the HEVC standard. Efficient implementations of these parallel options are described in [90] for slices, [127] for tiles and [35] for wavefront. Fine grained techniques usually consist of applying the Single Instruction Multiple Data (SIMD) paradigm at various levels of the encoding [8] and decoding stages [92].

As far as transcoding is concerned, a straightforward method is to first decode fully the input sequence, scale its resolution and then re-encode it. More efficient approaches target at utilizing the information already coded in the input, most noticeably the one concerning motion estimation, in order to reduce the search space when transcoding to another standard. Example works in the area include [60] where an H.264/AVC to HEVC transcoding architecture is presented that achieves a nominal speedup reaching 8x, when compared to re-encoding from scratch. If bitrate changes rather than a change in standard is needed, the process is often referred to as transrating. A survey on fast transrating methods can be found in [7]. In the experiments, as shown in the chapter 7.4 we obtained transcoding task weights by using the straightforward approach of re-encoding without using the information already coded. This was done both for reasons of simplicity and due to code availability (ffmpeg and x264 used). However, based on the aforementioned research we scaled the values obtained to depict the case where a more efficient transcoder is used.

Concerning Cloud transcoding, most works focused on providing job scheduling techniques at the level of a server cluster or a data center. In [13] the authors considered the case of live video transcoding and proposed an integer linear program (ILP) formulation to tackle scheduling decisions. An online algorithm that schedules jobs among the servers of a datacenter with the target of satisfying delay requirements while using minimum energy was proposed in [157]. In [102] the scope was a single cluster and the optimization target was to keep the servers load balanced. In [14] an admission control algorithm was developed that differs or rejects requests that can't be satisfied based on current workload. It is worth noting that this is the contrary approach to the one used in this chapter for the case of edges, whereby it might be viable to reduce quality by over-assigning tasks to servers if the relevant benefits from edge processing are deemed sufficient. Finally, in [62] a combined caching, transcoding approach is discussed, whereby transcoding jobs are partially processed to allow for efficient caching. The target considered in this chapter, i.e., live transcoding excludes partial transcoding as an option. Caching and replication techniques in the Cloud are surveyed in [105], while [79] and [80] concern efficient video delivery.

Overall, compared to [13], [157], [102], [14] and [62], we differ in scope since in section 7.3 we examine transcoding at edges while we view [105], [79] and [80] as orthogonal to our approach. Perhaps the closest work in the literature is [16], where system architecture for edge transcoding is described. Nevertheless, scheduling issues were not tackled in the manner done in this chapter.

6.3. Overview on Scheduling Video Transcoding Jobs over the Cloud

Video transcoding is the process of producing from an original input video sequence, multiple output sequences, each at potentially different bitrate, resolution and/or format. Transcoding is essential to support video delivery towards clients that use different players and have different network access capabilities. In the most basic scheme the input sequence is decoded and then re-encoded at the desired levels. Although significant research on fast transcoding schemes exists, the transcoding process is still computationally intensive. For this reason parallelization is typically used both at the core level of a single server and among the servers of a cluster. On top, efficient scheduling methods that allocate resources to transcoding jobs are necessary to achieve good overall performance. Such policies usually aim at allocating transcoding jobs over co-located servers, thus, they typically overlook parameters such as network traffic. Motivated by the case of transcoding in the Cloud, in chapter 8 we investigate the problem of scheduling transcoding jobs over a distributed system comprising of processing nodes that are geographically dispersed and might be whole clusters or even separate data centers. We propose algorithms to minimize both the inter-node network traffic and the intra-node energy consumption, while meeting the deadlines and quality requirements. Through simulation experiments we conclude on the best alternatives.

6.3.1. Energy Efficiency in Datacenters

Building energy efficient data centers has attracted much research effort recently, with the scope varying all the way from the case of optimizing single clusters to holistic data center design. In [141] energy efficient scheduling algorithms for heterogeneous clusters are proposed, while [96] and [99] characterize the energy efficiency of Hadoop clusters. In [30] a comparative study between 13 scheduling algorithms is done with an interest on energy consumption, while [19] focuses on comparison among DVFS techniques. Applications of DVFS but for Hadoop clusters are also studied (among others) in [81]. Finally, [158] discusses energy efficient in Hadoop clusters both from the standpoint of minimizing the total consumed energy and the peak power.

As far as data center level optimizations are concerned, [47] provides a thorough survey on modeling energy consumption, while [51] focuses on design principles for

energy efficiency with a special interest on data centers providing video related services. A key component to reducing data center power consumption is the successful implementation of server consolidation. Server consolidation is the process of maintaining in active state the minimum set of servers that meet workload demands, placing the rest in hibernating mode or completely turning them off in order to save energy. In order to do so, VMs are migrated from the servers that will be switched off to other destinations that will remain active. A plethora of algorithms was proposed that decide when and where to perform such migrations. Examples include [17] where online bin packing techniques are examined and [144] where process migration decisions are taken based both on server load but also on communication overhead. The interested reader is referred to [18] and [72] for related surveys.

In chapter 8.3 we don't aim to model the effects of particular strategies within a data center, but we are rather interested in proposing policies that are applicable regardless of the local strategies followed each time. As such, we don't assume that any particular mechanism for energy efficiency is implemented (DVFS, server consolidation). Nevertheless, the system model presented in 8.2 can be extended with minimum effort to capture such cases if required, while the global strategies proposed in that chapter remain unaffected.

6.3.2. Video Coding and Transcoding

A lot of research efforts were devoted into reducing the computational burden imposed by video codecs, usually by taking advantage of parallelism. Examples on the category include for instance [90], [117], [127], [8] and [92] each tackling parallelism at a different level. Slice level parallelism was studied for instance in [90] and [117] whereby a frame is split into independent regions (slices) that can be encoded separately and each slice is assigned to a separate CPU core. Tile level parallelism, another method to split a frame into independent regions introduced by High Efficiency Video Coding (HEVC), was discussed in [127]. These methods are rather coarse grained since they provide parallelism at the level of group of Macroblocks (H.264/AVC) or CTUs (blocks in HEVC). Finer grained approaches rely to SIMD instructions in order to parallelize the various components of the codec (motion estimation, compensation and transform are

usual candidates). Example works include [8] where SIMD parallelism is discussed for HEVC encoding and [92] where the focus was to parallelize the various parts of an AVS decoder.

The aforementioned works aim at speeding up the coding process when the input is a raw video sequence. Efficient transcoding solutions that operate over already coded sequences were also proposed. Examples include [60] where the transcoding from H.264/AVC to HEVC is discussed and [49] where an HEVC to VP9 transcoder is proposed. Research in this area, usually aims at reusing the prediction information of the standard the sequence is already coded in order to avoid the prediction overhead in the targeted standard. Thus, impressive speedups (of 7x, even more) can be achieved compared to the process of decoding completely the original sequence, obtain the raw file and then encoding it to the required target standard.

Since the focus of chapter 8 is not to optimize a particular transcoding task, but rather to schedule transcoding jobs efficiently at a higher level, we don't take any particular assumptions on the optimizations that take place at a video coding level. Furthermore, real world service providers might use their own codecs for which details are unavailable. In the experiments, we used the very popular `x264` [146] and `x265` [154] codecs for H.264/AVC and HEVC standards respectively, in order to obtain realistic values for our simulations.

6.3.3. Cloud Transcoding

Contrary to video delivery that has been the target of extensive research effort, see [150] for a recent survey, few works relatively exist on Cloud provisioning for video coding jobs. In [156] the parallelization of an encoding task over a Hadoop cluster is proposed at a Group of Pictures (GOP) level. GOPs are mapped to the various processing nodes and are independently encoded. The reduction phase includes synthesizing the final coded sequence from its sub-parts. Although this approach promises a high parallelization degree for a standalone video sequence, its performance is questionable in the presence of high load due to the overheads incurred with Map-Reduce. For this reason, in chapter 8 we assume that parallelization (if any) occurs within

a single server. In other words a transcoding task is executed by a single server and not multiple ones.

Closer to chapter 8 are the works in [157], [13] and [62]. In [157] the authors propose an online algorithm to decide about the server that will host a transcoding job. Parameters such as video resolution, server power and queue lengths are taken into account in order to optimize both processing delay and energy consumption. In [13] the case of live video streaming is tackled. The authors proposed an ILP formulation to decide about the transcoding rates of each stream that could be achieved based on the available processing capacity, having as a final goal the optimization of user experience.

Chapters' 8 work differs from the aforementioned papers in scope. Namely, whereas the focus of related work was on optimization at the level of a single cluster or at best on a single data center, we aim at investigating the performance of global scheduling schemes that distribute jobs over different distributed data centers. That said, we use the same dataset as in [13] to simulate the case of live transcoding jobs.

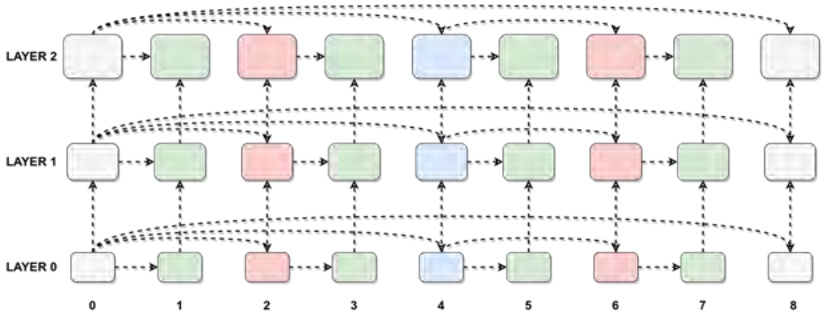


Figure 6-1 Spatial-time representation of video coding in layers according to SVC.

In [62] the focus is to decide on which video parts to transcode and cache them, based on the observation that users seldom view a video sequence completely. This work is mostly applicable to service providers offering video downloading, e.g., YouTube. In the chapter we target at developing a unified scheduler to be used by services offering both live (e.g., Wowza Streaming Cloud) and video file transcoding (e.g., Amazon Elastic Transcoder). In both scenarios the possibility of partial transcodings doesn't exist.

Last but not least we would like to mention that an alternative to transcoding is provided by SVC (Scalable Video Coding) [123], whereby a video sequence is encoded in multiple layers. The base layer accounts for the minimal supported quality, while the other layers, called enhancement layers, encode additional information in order to account for increased quality. Figure 6-1 provides a graphical representation of this with 3 layers. As it can be viewed by the dependencies showed with directed edges, in order to decode layer 2, layers 1 and 0 are needed. Compared to transcoding, SVC doesn't require multiple files to be maintained, instead a single stream is sent and each decoder decides up to which layer it will decode (depending on the required quality). Despite its potential, SVC doesn't account for transcoding between different standards, while it also requires extensive coordination overhead. Therefore in chapter 8 we focus on transcoding jobs that account for the majority of related Cloud services.

7. Video Transcoding on Cloud Edges

7.1. Motivation

Modern applications built on top of an integrated Internet of Things (IoT) environment [71], together with Cyber Physical Systems (CPSs) [87], involve heavy video traffic, e.g., in smart vehicle traffic management. At the same time, the proliferation of smart mobile devices carrying cameras of continuously higher resolution, together with the explosive growth in the popularity of social media platforms, poses great challenges in cloud resource management. As an indication, Cisco reported in [40] that during 2015, mobile Internet traffic experienced a growth of 74%, the majority of which (>50%) was video transmissions. Therefore, minimizing video related network traffic becomes of paramount importance.

Video coding is the process of compressing a raw video sequence using some standards. Examples of such standards are H.264/AVC [152] which is the most popular (but aging) standard currently in use, High Efficiency Video Coding (HEVC) [133] and VP9 [70], which are newer standards achieving higher compression ratios compared to H.264/AVC. Although video coding is a computationally demanding task, it is usually performed at the point where the initial video is captured (camera, smart device etc.), often with the aid of specialized hardware. Thus, the initial coding of a video sequence does not hinder a cloud based social media platform (SMP) from being computationally wise and the only overhead is the consumed bandwidth for uploading. However, in order to be able to deliver the video sequence to a variety of clients differing in screen resolutions, decoders and network capabilities, the originally uploaded sequence must be encoded into multiple output sequences of various resolutions, bitrates, quality levels and perhaps coding standards. This process is called transcoding and burdens the computational and network-wise SMP's cloud. In particular, the case of live casting offers the most challenges since real time performance is a requirement.

Motivated by the above, we investigate the case where an SMP can take advantage of mini-data centers existing at network edges in order to offload live transcoding jobs, thus, saving resources and bandwidth. Figure 7-1 illustrates an example whereby two broadcasts are performed, one at 1080p and the other at 720p from two different edges.

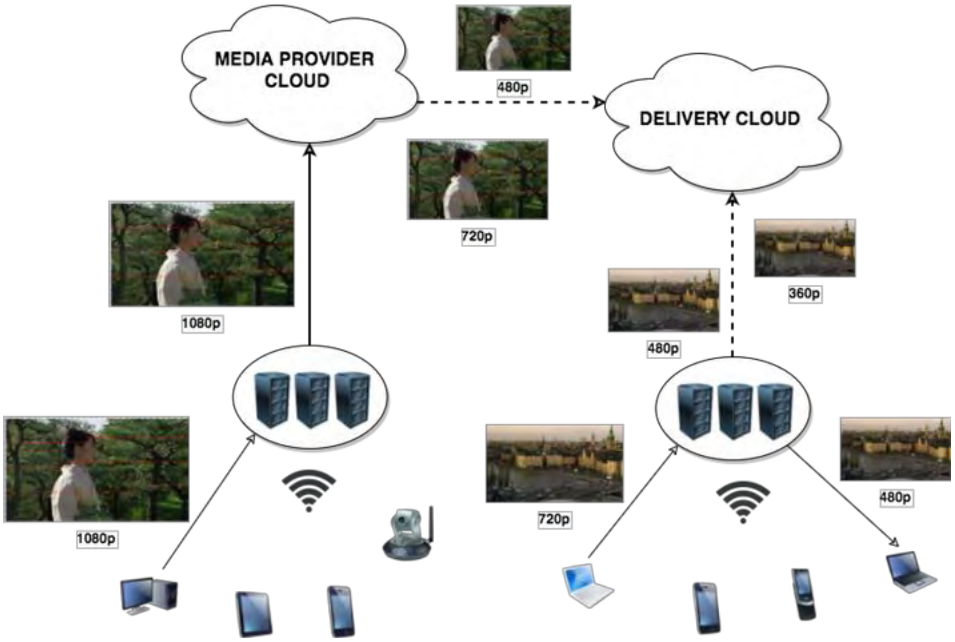


Figure 7-1 Example system model with edge transcoding

In the first case (1080p), the sequence is not transcoded at the edge but transmitted to one of the SMP's data centers for processing. Then two different outputs (720p and 480p) are sent to some Content Delivery Network (CDN). In contrast to this, the other input sequence (720p) is transcoded into two output sequences at the edge. Copies of the outputs are sent to the CDN and also used to satisfy local demands (480p). Clearly, the second alternative of using edge transcoding reduces both the processing and network resource consumption at the SMP's Cloud.

In this chapter, we tackle the associated scheduling problem induced by the scenario of Figure 7-1. Namely, given edge resources and the characteristics of arriving transcoding tasks, task-server assignment must be made so that the percentage of tasks not processed by the edge (satisfied with overhead by SMP's Cloud) is minimized. We evaluate different scheduling heuristics for the scheduling problem under the constraint that each assigned task must obtain the required processing power to exhibit real time behavior. We then examine the case where the aforementioned constraint is softened,

allowing for some quality loss in order to increase the number of tasks assigned to the edge. All heuristics are evaluated using a dataset of Twitch broadcasts [13] and realistic values for transcoding job characteristics obtained by using x264 codec [146] over class B and class A common test video sequences [21].

7.2. Problem definition

We consider the case of a media provider receiving requests for live video casting, whereby the input stream must be transcoded into a set of output streams with different resolution, bitrate and quality demands. We consider two options for the set of transcoding tasks associated with each input. Either they are all assigned to a mini-datacenter existing at the edge of the network or they are all assigned to the backend main datacenter of the media provider. Clearly, if the tasks are processed at the edge, the processing workload at the backend datacenter is reduced and the network overhead for transmitting the input sequence is avoided.

Let the mini-datacenter consist of S servers, with S_i denoting the ith of them, assuming a total ordering ($1 \leq i \leq S$). Each server has an associated processing capacity (let C_i), which denotes the number of baseline transcoding tasks that can be processed concurrently at real time. Baseline tasks are the ones requiring the minimum power to process. Let B_j be the jth broadcast, assuming an ordering of the B total broadcasting events ($1 \leq j \leq B$). Similarly, let s_j and d_j be the arrival time and duration of B_j, respectively. Each broadcast entails a set of transcoding tasks. Let T be the total number of transcoding tasks for all broadcasts, and T_k be the kth such task, assuming a total ordering of them ($1 \leq k \leq T$). We represent whether T_k is a task of B_j or not, using a Boolean matrix A of $B \times T$ size, whereby $A_{jk}=1$ if and only if (iff) B_j has task T_k and 0 otherwise. Moreover, W_k depicts the relevant weight of T_k in processing terms over the baseline task. Put in other terms, W_k shows how much more computationally demanding T_k is, compared to the baseline scenario. Last, let X be an $S \times T$ Boolean matrix used to encode task server assignments as follows: $X_{ik}=1$ iff T_k is assigned for processing at S_i, otherwise $X_{ik}=0$. We assume that once assigned, a task cannot be preempted and will remain for the whole duration $[s_j, .., s_j+d_j]$. We consider that we want to optimize the system starting from a clean state (no task assignments exist) over a time frame divided

into E equally sized slots (s_j and d_j values are now measured in time slot terms). Let e_t be the tth such time slot, with a corresponding assignment matrix X^t. We typically formulate the problem as follows: Find all values in the E total matrices X^t, so that the objective function f given in (7.1) is maximized:

$$f = \sum_{t=1}^{E} \sum_{i=1}^{S} \sum_{k=1}^{T} X_{ik}^{t} \, (1 - X_{ik}^{t-1}), \tag{7.1}$$

subject to the following constraints:

$$\left(\sum_{i=1}^{S} \sum_{k=1}^{T} A_{jk} \, X_{ik}^{t} - \sum_{k=1}^{T} A_{jk} \right) \sum_{i=1}^{S} \sum_{k=1}^{T} A_{jk} \, X_{ik}^{t} = 0, \quad \forall j, \; t = s_j, \tag{7.2}$$

$$X_{ik}^{t} = X_{ik}^{t+1}, \quad \forall i, k, \; t | s_j \leq t < s_j + d_j \wedge A_{jk} = 1, \tag{7.3}$$

$$\sum_{i=1}^{S} \sum_{k=1}^{T} A_{jk} \, X_{ik}^{t} = 0, \quad \forall j, t | t < s_j \vee t > s_j + d_j, \tag{7.4}$$

$$\sum_{k=1}^{T} X_{ik}^{t} W_k \leq C_i, \quad \forall i, t, \tag{8.5}$$

$$\sum_{i=1}^{S} X_{ik}^{t} \leq 1, \quad \forall k, t. \tag{7.6}$$

The objective function encodes the tasks that will be assigned to the edge. Eqs. (7.2)–(7.6) give the main constraints of the problem. Constraint (7.2) states that either all tasks of a broadcast B_j will be assigned to the edge at the time the broadcast arrives or none. Constraint (7.3) enforces that the decision taken for a transcoding task at the time of its broadcast arrival remains for the duration of the broadcast. Constraint (7.4) ensures that neither before a broadcast arrival, nor after its end time, can a corresponding task be scheduled for edge transcoding. Constraint (7.5) dictates that a server can exceed its capacity at no point in time. Finally, (7.6) states that a task can only be scheduled at one server.

Clearly, the fact that broadcasts are known in advance reduces the applicability of the presented problem formulation to cases of prescheduled event covering, e.g., sports. Nevertheless, the formulation provides a thorough definition of the optimization target and the related constraints. These remain the same both in the static problem variation presented and in the dynamic case. A last note concerns complexity. It can be shown that the relevant decision problem is NP-complete since the processing capacity constraint at the servers effectively introduces a (0, 1) Knapsack component. Next, we present heuristics for dynamic scheduling of transcoding tasks at the network edge.

7.3. Scheduling heuristics

7.3.1. Scheduling with Tight Task QoS Requirements

The proposed heuristics tackle the dynamic version of the scheduling problem presented in the previous section. Specifically, upon the arrival of a broadcast request, the necessary transcoding tasks are defined. Then, they are sorted according to their weight and considered either in increasing order (MIN policy) or in decreasing (MAX policy). Each task is assigned to a server (using one of the policies described in the sequel) provided the task computational demands can be met by the server as per (7.5). If a suitable server is found for every transcoding task of the broadcast under consideration, the assignments are committed; otherwise, even if one task fails to find a hosting server, all the tasks are sent to the SMP's datacenter for processing. The assignment policies considered are based on the well-known bin-packing heuristics:

- Best Fit (BF): Select the server where the remaining capacity, left after task assignment, is the minimum possible.

- Worst Fit (WF): Similar to BF only that the server with the maximum remaining capacity will be selected.

- First Fit (FF): The first server where the task fits will be selected.

The corresponding heuristics are named after the order with which the task list is considered and the packing method followed. For instance MAX-BF refers to the heuristic that considers the heaviest task first and assigns it using Best Fit.

7.3.2. Scheduling with Relaxed Task QoS Requirements

The motivation for the relaxed QoS case is the following. Assume that all but one task of a broadcast could fit to the available servers of the edge. With strict QoS requirements, none of these tasks will be assigned. However, it might be possible to assign the remaining task to one server so that its processing capacity is exceeded by a very small margin. In practice, this means that all the tasks processed by this server will exhibit a small quality drop. For instance, if a broadcaster transmits at 30 fps (frames per second) then a 3.3% drop at the processing rate of one of its transcoding tasks means that roughly the output stream will be at 29 fps. Depending on decoder characteristics, such a drop might not even be noticeable by a human viewer. Assuming that p denotes the maximum percentage of allowable performance drop, (7.5) becomes:

$$\sum_{k=1}^{T} X_{ik}^{t} W_k \leq (1+p) C_i, \quad \forall i, t. \tag{7.7}$$

The heuristics first attempt to allocate all the tasks of a broadcast as per Section 7.3.1. In case a task does not fit, it is considered for assignment using (7.7) as server capacity constraint and one of the below described policies.

- Min Quality Decrease (MQD): Selects the server that incurs the minimum proportional capacity violation (equivalent to asking for the minimum quality penalty for its hosted tasks).

- First Fit (FF): The first server where the task fits as per (7.7) will be selected.

- View Weighted Penalty (VWP): Weights the quality penalty of each task by the number of its viewers. The server with the minimum aggregated weighted quality penalty value is selected.

7.4. Experiments

Here we illustrate our simulation results. We begin by presenting the experimental setup and proceed with the performance of the heuristics under the strict and soft quality requirement scenarios.

7.4.1. Setup

To simulate broadcasting activity, we used the same dataset from Twitch as the one described in [13]. We kept the portion of the dataset representing one day activity (Jan. 6th, 2014). We then filtered it by deleting entries with broadcasts having no viewers and the broadcasts of resolution less than 220p. To keep the simulation time manageable we considered the following 5 resolutions: 240p, 360p, 480p, 720p and 1080p. In case a broadcast in the trace did not follow one of the previously mentioned resolutions, we clustered it to its closest matching. We assumed that a broadcast must be transcoded to all the resolutions that were lower than the one it used. Clearly, with this setting the maximum number of transcoding tasks incurred by a broadcast is 4, corresponding to a 1080p stream that must be downscaled to 720p, 480p, 360p and 240p. Upscaling was not considered in the experiments. Finally, for simulation purposes we assumed that all videos used 30 fps. Furthermore, the recorded in the dataset viewing demand was split equally among the resolutions used by a broadcast, i.e., the input and all lower ones. Table 7-1 summarizes some of the dataset characteristics, while Figure 7-2 plots the broadcasting job arrival rates as a histogram of a 1000 seconds (s) step. As it can be seen, the arriving jobs do not exhibit sharp peaks (at least with the used interval), but the distribution is rather uniform. This favors job scheduling at edges since it makes sizing decisions for edges less demanding. However, duration of broadcasts does not follow a similar trend. As noted in Table 7-1, the difference between the average and maximum duration is two orders of magnitude, implying a heavy tailed distribution. This hinders scheduling decisions, since it means that duration estimation will be hard to achieve in the general case. For this reason, none of the scheduling heuristics described in Section 7.3 uses such estimates.

Next, we needed to characterize the weights of the transcoding tasks. For this reason we used class A and class B common test video sequences and transcoded them to the levels for which we wanted to obtain weight values. To do so, a sequence was first fully decoded, then scaled to the desired resolution using ffmpeg and then encoded using x264. The encoding settings followed the Peak Signal to Noise Ratio (PSNR) tailored scenario of [48], which aims at maximizing quality in PSNR terms.

Name	Value
Dataset duration	75,079 s
Average broadcast duration	23,263.4 s
Max broadcast duration	1.05E6 s
Number of broadcasts	786,100
Total transcoding tasks	1,244,450
Percentage of braodcasts at 1080p	26.21%
Percentage of braodcasts at 720p	53.24%
Percentage of braodcasts at 480p	11.18%
Percentage of braodcasts at 360p	7.49%
Percentage of braodcasts at 240p	1.86%

Table 7-1 Dataset for broadcasters (general characteristics)

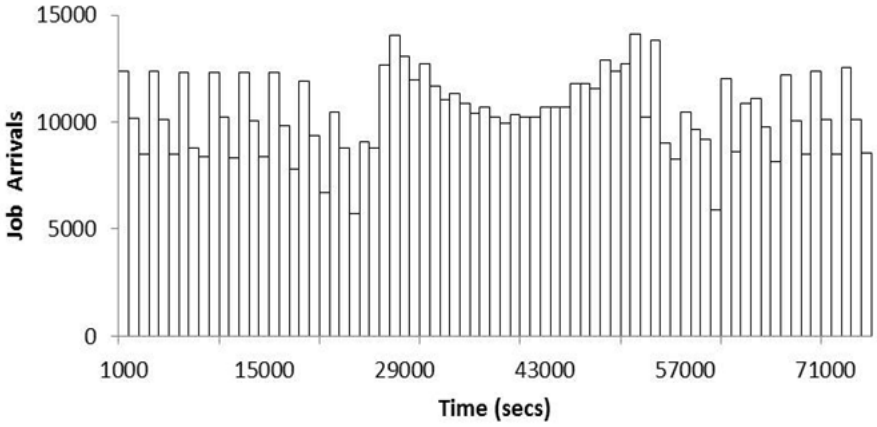


Figure 7-2 Histogram for broadcasting arrival rates.

Name	Resolution	Frames	Time 240p (fps)	Time 360p (fps)	Time 480p (fps)	Time 720p (fps)
BasketballDrive	1920×1080	500	15.78	5.37	3.78	1.47
BQTerrace	1920×1080	600	29.10	9.11	6.42	2.37
Cactus	1920×1080	500	25.48	8.77	5.09	1.96
Kimono	1920×1080	240	18.54	6.05	4.33	1.62
ParkScene	1920×1080	240	24.95	7.62	5.28	1.94
PeopleOnStreet	2560×1600	150	10.01	2.82	2.00	0.74
Traffic	2560×1600	150	27.36	8.06	5.77	2.17
Average	-	-	21.60	6.82	4.66	1.75
Weights	-	-	1.00	3.16	4.63	12.34

Table 7-2 Video sequences used for weight calculation

The exact parameters are given below (Kimono example): x264 --input-depth 8 --frames 0 --input-res 1920x1080 --fps 24 --input-csp i420 --log-level debug --tune psnr --psnr --profile high --preset placebo --keyint 96 --min-keyint 96 --me umh --merange 240 --ref 4 --partitions all --threads 1 --subme 9 --aq-mode 0 --aq-strength 0.0 --psy-rd 0.0 --output kimono_out.264 Kimono_in.yuv.

Table 7-2 summarizes the general characteristics of the test sequences, together with the coding time for each targeted resolution, measured as the number of frames per second processed by the codec. The case of 240p forms the baseline transcoding scenario, with the remaining resolutions assigned proportional weights.

Having defined the time of the baseline scenario and task weights accordingly, next we define server capacity in the following manner. In order to fully control the system environment we used a dedicated server for which we had full ownership. The server used for the x264 coding jobs carried two 6-core Intel Xeon E5-2630 CPUs running at 2.3 GHz. Since the coding speeds at Table 7-2 used one thread and the nominal rate considered for the simulation is 30fps, each core of the server accounts for a processing capacity of 21.6/30 (the baseline scenario). The total server capacity is then calculated by multiplying with 12 (the total number of physical cores) and equals 8.64.

Since our server setting is not of generic use, we translate it into one of the Amazon EC2 instances [12] to make our simulation setting more applicable. Specifically, we consider the C3 instances which are recommended for video coding. Comparing the processor passmark ratios between the CPU of our server and the one used in the C3 instances, i.e., Intel Xeon E5-2680 v2 (Ivy Bridge), it can be estimated that the instance c3.4xlarge will account for a speedup of 2.2x compared to our server. Since video coding is CPU-bound, the aforementioned methodology (i.e., comparing CPUs) provides a good estimation on relative performance. Last, we consider the case where specialized transcoding software is available, which accounts for a speedup of 8x (same as in [60]) compared to the simple methodology used to obtain the processing rates of Table 7-2.

7.4.2. Results for Tight QoS Requirements

Here we present results for the case where the assigned transcoding tasks at the edge must be satisfied at their nominal rate (30 fps). We consider two scenarios. In the first (homogeneous), 1000 servers each of capacity described in Section 7.4.1 exist in the micro datacenter of the edge, while in the second scenario (heterogeneous) 500 servers have the aforementioned capacity and 500 half of it (presumably equivalent to a c3.2xlarge EC2 instance). Figure 7-3 plots the performance of the scheduling heuristics, measured in terms of the percentage of broadcasts that are assigned for edge transcoding. Results show that the same trends are exhibited both in the homogeneous and the heterogeneous cases. The later achieves lower performance since it accounts for smaller total capacity. Furthermore, sorting the transcoding tasks of a broadcast has marginal effect. This is presumable due to the fact that the tasks are scheduled as soon as a broadcast arrives and are rather small in number, compared to the available servers. Clearly the WF policy outperforms the other alternatives by a substantially large margin (an extra 5% roughly of the arriving requests can be accommodated by the edge).

Having identified WF to be the most promising heuristic, we evaluated the impact of the arrival rate on the achievable performance. To do so, we used the same trace with above, but sampled it every 1, 2 and 3 entries. Clearly, a sampling rate of 1 is equivalent

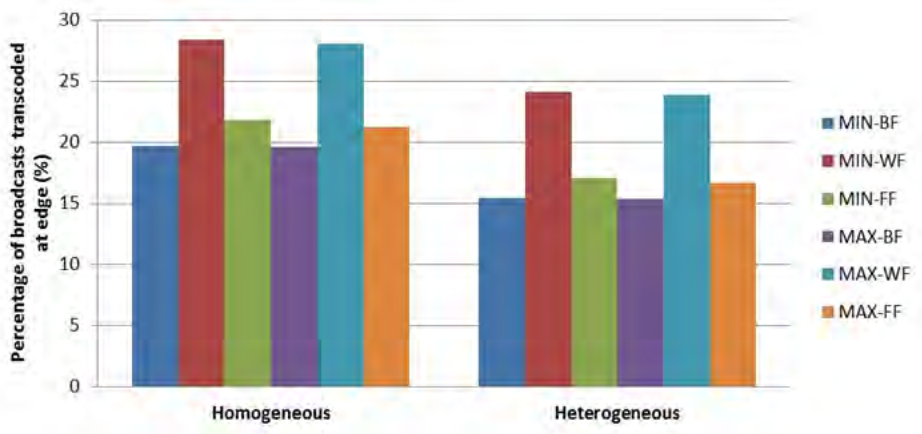
to using the whole dataset (Figure 7-3), while 2 and 3 effectively account for 1/2 and 1/3 of the arrival rate. Figure 7-4 plots the performance of the WF scheme for the three arrival rates and for both the homogeneous and heterogeneous cases. As expected, the percentage of jobs that can be satisfied by the edge increases as the arrival rate decreases. It is worth noting that with 1/3 arrival rate which accounts for roughly 262,000 daily requests, roughly 60% of them (homogeneous case) can be satisfied by the edge. This translates for great load reduction at the back end datacenters.

7.4.3. Results for Soft QoS Requirements

We consider that the processing requirement of real time performance is relaxed as per Section 7.3.2. We evaluated the performance of the algorithms of Section 7.3.2 when combined with MAX-WF for various levels of QoS reduction, namely: 0% (no reduction-real time performance), 5%, 10%, 15% and 20%. Figure 7-5 plots the percentage of broadcasts having their transcoding jobs assigned to the micro datacenter, for the heterogeneous case used in Figure 7-3 and Figure 7-4 and with the whole trace as input. All three heuristics assign to the edge an increasing number of jobs as QoS requirements are reduced. Among the algorithms, MQD achieves the best performance, followed by VWP and FF. As it can be observed, with a 5% decrease in processing rate, an extra 2% of jobs can be assigned to the edge, while with 20% an extra 4.5%. Although a 20% reduction seems impractical at first glance, it roughly means that instead of processing a stream at 30 fps, the stream might be processed at 24 fps. It is worth noting that this is the lowest rate for 1080p TV sets. Overall, by relaxing the nominal real time processing rate for transcoding jobs, significant extra load could be offset from back end datacenters.

In order to further quantify the impact of QoS reduction, in Figure 7-6 we plot the average viewing quality (for the viewers satisfied by the edge) as a percentage of the achieved fps when compared to real time 30 fps. VWP achieves the best performance, which for a 15% allowable reduction (0.15 point in x-axis) leads to an average viewing quality of 93%. Put it in other terms the average processing rate will be almost 28 fps. For the same QoS reduction (15%), Figure 7-5 depicts that VWP assigns an extra ~3.5% of jobs, or roughly 27,000 more broadcasts at the edge micro datacenter. Thus,

an interesting tradeoff is present whereby VWP can offset substantial load towards the edge micro datacenter at only a small decrease on average viewing quality.



BF: Best Fit
FF: First Fit
WF: Worst Fit

Figure 7-3 Percentage of broadcasts processed by the edge (1000 servers, full dataset). Two different cases: homogeneous and heterogeneous.

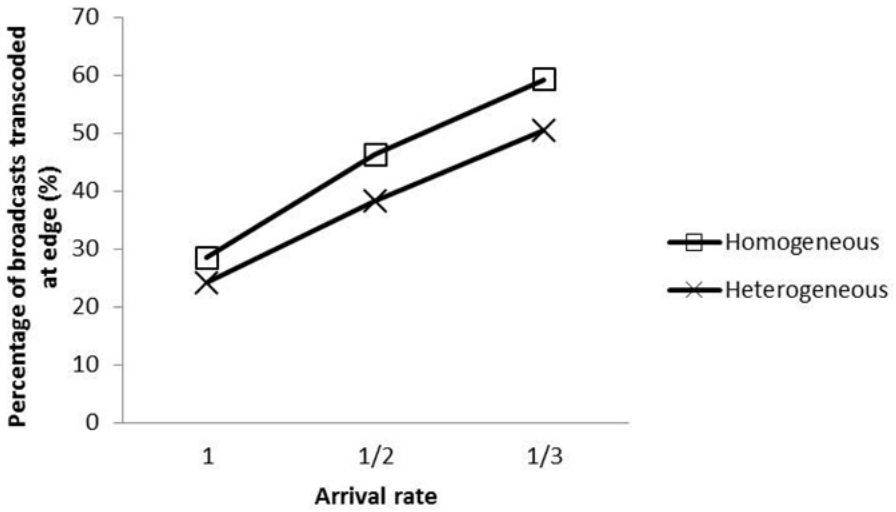


Figure 7-4 Percentage of broadcasts processed by the edge for decreasing arrival rate (1000 servers).

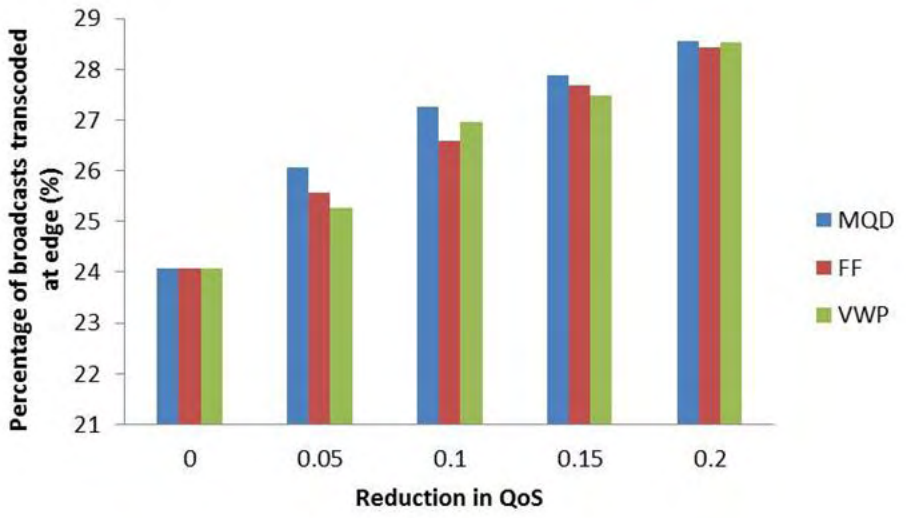


Figure 7-5 Percentage of broadcasts processed by the edge for varying QoS reduction percentages (full dataset, heterogeneous servers).

FF: First Fit
MQD: Min Quality Decrease
VWP: View Weighted Penalty

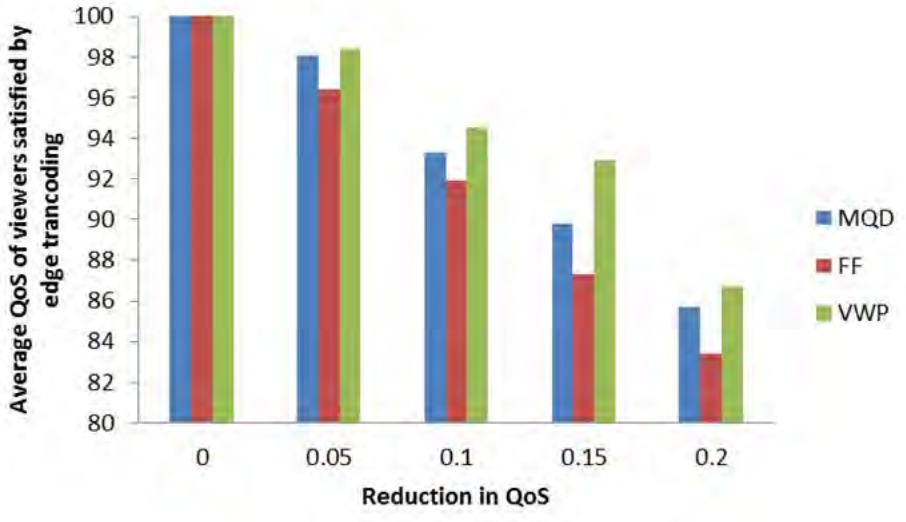


Figure 7-6 Average QoS of viewers as the allowable reduction in QoS for edge transcoding jobs is increased (full dataset, heterogeneous servers).

8. Scheduling Video Transcoding Jobs over the Cloud

8.1. Motivation

Video uploading and downloading is one of the most popular activities among social media users. Cisco reported in [40] that in 2015 a year whereby mobile Internet traffic experienced a rise by 74%, 55% of it accounted for video. These trends will likely continue, even intensify, as modern smart devices with 4K cameras are established in the market and 8K TV screens replace older ones. To cope with an everlasting demand for higher resolution, new video coding standards such as VP9 [49] and HEVC [133] were introduced in recent years, while the quest for future generation standards have already begun, e.g., AV1 [9], JVET [61]. All these new standards aim at replacing the long standing H.264/AVC standard [152], by offering increased compression ratios without sacrificing quality.

At the same time, the delivery of video streams towards end devices of different screen and processing capabilities that use different players and reside on networks of various bandwidth capacities, poses a major challenge. To cope with the problem, transcoding is used whereby from the initial video sequence, a set of output sequences is produced, each potentially at different resolution, transmission rate and quality level. On top, transcoding might also involve the change of the coding standard, e.g., from H.264/AVC to HEVC [90]. The various rates and qualities on which an original sequence is transcoded (also called the encoding ladder) can be of large number. For instance, in Netflix a ladder of at least 10 different levels (probably more) is advocated in [1]. It is evident that even with such a conservative ladder, the processing demands posed by the transcoding needs of large media providers can't be met without taking advantage of Cloud resources. For this reason a number of Cloud service providers, include nowadays video coding services, e.g., Amazon Elastic Transcoder [11], or transcoding services as part of their live casting service, e.g., Wowza Streaming Cloud [141].

In this chapter we consider the case of a Cloud service provider offering video coding and transcoding services and tackle the related scheduling problem. In particular we discuss the case whereby the provider owns a number of geographically distributed data

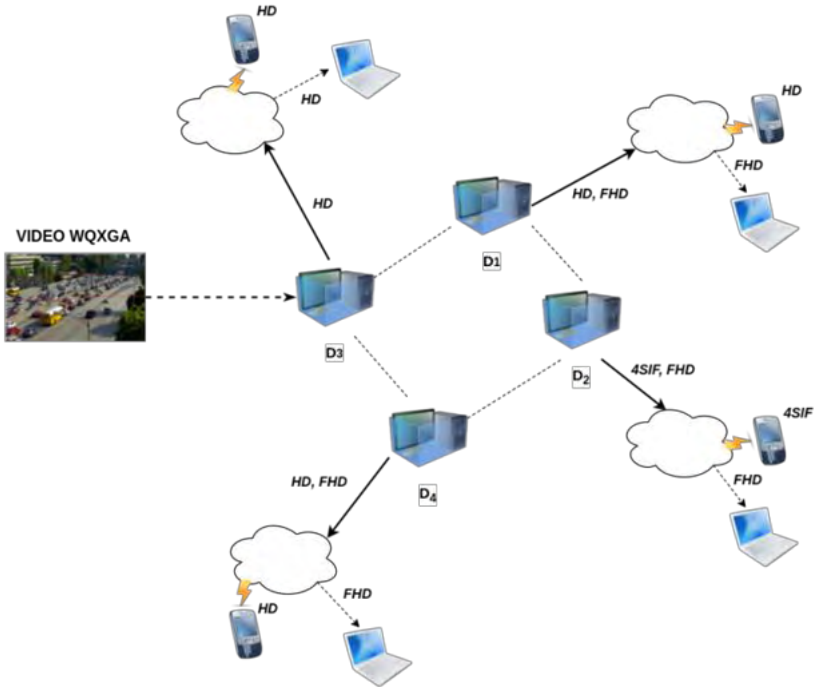


Figure 8-1 An example of a Cloud transcoding service

centers and coding jobs arrive from different locations. An example of such a system is given in Figure 8-1 that shows an arriving video at data center D3, being transcoded at various targeted resolutions and delivered in multiple end points. We focus on investigating global scheduling approaches whereby it is decided in which data center to assign each transcoding job so as to optimize completion time, energy efficiency and network traffic. Our contributions include the following:

- We use a system model that captures the practical case of interest of Cloud transcoding service providers operating over different data centers. Furthermore, the model and subsequently the algorithms, capture both the case of live transcoding tasks and the case of offline video sequence coding;

- We develop dynamic global scheduling algorithms that have different optimization goals, i.e., processing time, energy and network overhead, and incur minimum coordination overhead with the datacenters involved;

- We also propose a scheme that involves closer cooperation between the global scheduler and the local schedulers at datacenters, and investigate the most promising trade-offs between solution quality and coordination overhead;

- We conducted two sets of simulation experiments, one tailored to capture the system's behavior in live casting scenarios and the other one the system's performance when a large set of H.264/AVC sequences must be transcoded into HEVC;

Results indicate that a global scheduler that decides based on pareto optimality upon the three parameters: time, energy, network offers the best approach, while incurring small coordination overhead.

8.2. System Model

We assume a Cloud transcoding service with computational resources spanning across different geographically distributed data centers. In order to facilitate description throughout this section we introduce some notations that are summarized in Table 8-1. Let D be the total number of available data centers and D_i denote the i^{th} such assuming a total ordering of them ($1 \leq i \leq D$). Each data center contains a number of servers. Such servers might in fact be virtual machines running over physical ones. As already mentioned we refrained from tackling the implementation particulars of data centers, e.g., VM allocation and migration policies. Therefore, for all purposes, the servers are assumed to be dedicated for transcoding jobs only. Let S be the total number of servers across all the available data centers dedicated for transcoding. We denote by S_k the k^{th} such server assuming a total ordering of them ($1 \leq k \leq S$).

Each server potentially has its own characteristics concerning processing capacity and power consumption. We denote server processing capacity by Cj. It is straightforward that this capacity can be measured as CPU speed, FLOPS or any other similar metric in the literature. However, in order to both simplify the model but also be able to capture the required performance characteristics without introducing additional parameters such

Symbol	Meaning
D	Total number of data centers
D_i	The i^{th} data center
S	Total number of servers in all D data centers
S_k	The k^{th} server
C_k	Processing capacity of S_k
PW_k	Power consumption of S_k
E_k	Energy efficiency of S_k
h_{ij}	Distance between D_i and D_j (in hops)
J	Total number of jobs
J_x	The x^{th} job
JT_x	The number of tasks associated with J_x
T_{xy}	The y^{th} task of J_x
W_{xy}	Processing load incurred by T_{xy}
l_x	Time length of input file/stream of J_x
$b(J_x)$	Total size length of input file/stream of J_x
$b(T_{xy})$	Total size length of output file/stream of T_{xy}
q_{xyk}	Service quality of live transcoding T_{xy} at S_k
AJ_x	Arrival time of J_x
ET_{xyk}	End time of T_{xy} assigned to S_k
A_{ix}	Boolean variable denoting whether J_x first arrives at D_i
O_{ixy}	Boolean variable denoting whether the output of T_{xy} is required by end users in the
P_{xyk}	Boolean variable denoting whether T_{xy} is assigned for execution in S_k
IN_{ix}	Boolean denoting whether D_i must get the input stream of J_x
B_{ik}	Boolean variable denoting whether S_k belongs to D_i.

Table 8-1 Notation Used in this Chapter

as memory or number of cores, we measure processing capacity as the number of baseline video sequences that can be handled concurrently by the server. The baseline is assumed to be a raw 240p video that must be encoded in 30 frames per second using H.264/AVC (the particular coding settings used are detailed in 8.4).

Each server is also characterized by its energy efficiency (let E_k). As already stated we don't consider DVFS since we focus on a higher than a cluster scheduling level. Instead we measure energy efficiency at each server based on its maximum processing capacity.

Assuming PW_k to be the corresponding power consumption rate (Watts) the energy efficiency of each server is given by:

$$E_k = C_k / PW_k \tag{8.1}$$

We assume that the data centers are connected with high speed network links, effectively forming a network graph. Among others we are interested in reducing the network overhead due to transcoding jobs. We measure this overhead as the mere amount of bytes passing through each link (totaled for all links). We decided to follow a conservative estimation whereby the communication between D_i and D_j always occurs along the shortest path of h_{ij} hops.

Coding job requests arrive (presumably at some edge point) and are directed towards the closest data center. Let J_x denote the x^{th} coding job assuming a total order over the J total jobs that exist. Depending on whether the job represents a standalone file transcoding or a live casting event, it might result into one or more transcoding tasks respectively. Let JT_x be the number of transoding tasks associated with J_x and T_{xy} be the y^{th} such transoding task assuming a total ordering of them ($1 \leq y \leq JT_x$).

All the tasks of J_x are associated with the same input file or stream but with different outputs. Depending on the type (raw, H.264/AVC etc.) and the resolution of the input file, as well as the resolution and the type of the required output by a particular task, different processing capacity is required to achieve real time performance (e.g., 30 fps). Let W_{xy} denote the processing capacity required by T_{xy} measured as the ratio of load incurred by this particular transcoding versus the baseline case. Both the input sequence of J_x and all output sequences of the associated tasks have the same length (in secs), denoted by l_x. Nevertheless, they have different total sizes (in bytes). Let $b(J_x)$ denote the size of the input in J_x and $b(T_{xy})$ the size of the output of T_{xy}.

Assuming T_{xy} is assigned to an initially empty S_k, we distinguish two cases: (a) T_{xy} corresponds to live transcoding in which case for a duration of l_x, and assuming $W_{xy} \leq C_k$, W_{xy} out of C_k total processing capacity will be reserved by T_{xy}, leaving a remaining

capacity of C_k-W_{xy} to be assigned to other tasks. Also we will denote that the quality (let q_{xyk}) at which T_{xy} was served equaled 1. In case $W_{xy} > C_k$ all the server capacity will be occupied and the corresponding service quality will be:

$$q_{xyk} = W_{xy}/C_k \tag{8.2}$$

(b) T_{xy} is a standalone file transcoding in which case all C_k will be reserved and T_{xy} that arrived at time AJ_x (all the tasks of a job are assumed to arrive at the same time) will have an end time (let ET_{xyk}) equaling:

$$ET_{xyk} = AJ_x + W_{xy}l_x/C_k \tag{8.3}$$

We should notice that (8.2) and (8.3) hold for tasks arriving at empty servers. In case a server is assigned more than one task then it is assumed that its processing power C_k is evenly split among the hosted tasks.

An arriving request for J_x might not necessarily be satisfied by the data center it first arrives. Furthermore, it isn't necessary that all its tasks will be assigned to the same data center for processing. To calculate the network overhead incurred by J_x one must add the overhead for fetching the input stream to the position(s) where task processing will occur and the overhead for sending the outputs of the transcoding tasks to the necessary destinations.

Let A_{ix} be a boolean variable, with A_{ix}=1 iff J_x first arrives in D_i and 0 otherwise. Let O_{ixy} be a boolean variable, whereby O_{ixy}=1 iff the output sequence of T_{xy} must be sent to satisfy viewers that are located in the geographic neighbor of D_i and 0 otherwise. Let P_{xyk} be a boolean variable depicting whether T_{xy} is assigned to S_k (P_{xyk}=1) or not (P_{xyk}=0). Furthermore, let IN_{ix}=1, if the input stream of J_x must be made available at D_i and 0 otherwise. Obviously, IN_{ix}=1 iff at least one of the tasks of J_x are assigned for processing at a server located in D_i. Last, let B_{ik}=1 iff S_k belongs to D_i and 0 otherwise. Then we can typically define the network overhead (let N_x) incurred by J_x as:

$$N1_x = \sum_{i=1}^{D}\sum_{j=1}^{D} A_{ix}\, IN_{jx} h_{ij}\, b(J_x) \tag{8.4}$$

$$N2_x = \sum_{i=1}^{D}\sum_{j=1}^{D}\sum_{y=1}^{JT_x}\sum_{k=1}^{S} P_{xyk} B_{ik}\, O_{jxy} h_{ij}\, b(T_{xy}) \tag{8.5}$$

$$N_x = N1_x + N2_x \tag{8.6}$$

subject to the constraints:

$$\left(1 - IN_{jx}\right)\sum_{y=1}^{JT_x}\sum_{k=1}^{S} P_{xyk} B_{jk} = 0 \quad \forall 1 \le j \le D \tag{8.7}$$

$$\sum_{y=1}^{JT_x}\sum_{k=1}^{S} P_{xyk} B_{jk} \ge IN_{jx} \quad \forall 1 \le j \le D \tag{8.8}$$

In (8.4) the overhead incurred by the input stream is captured; in (8.5) the overhead of all output streams, while (8.6) gives the total overhead for J_x as the summation of the two. (8.7) and (8.8) are necessary to ensure IN_{jx} takes valid values. Specifically (8.7) states that if at least one task of J_x is assigned for processing to one of the servers of D_j then IN_{jx} will equal 1. (8.8) forbids the case of having IN_{jx}=1 without assigning any of J_x's tasks at D_j.

We further illustrate network overhead calculation through Figure 8-2 which is similar to Figure 8-1 but simplified. The figure shows a transcoding job arriving at D_3 that consists of two tasks, one being to transcode from WQXGA to FullHD and another one to HD. Assume for the sake of the example that WQXGA size is b bytes, FullHD output is $b/2$ and HD $b/4$. By assigning the FullHD transcoding to be performed by D_2 and the HD transcoding by D_1, the network overhead is calculated as follows. The input overhead will be $3b$ ($2b$ to send to D_2 and b to send to D_1). The output overhead will be 0 for the transcoding of HD (only needed by the region of D_1) and $b/2$ for the FullHD

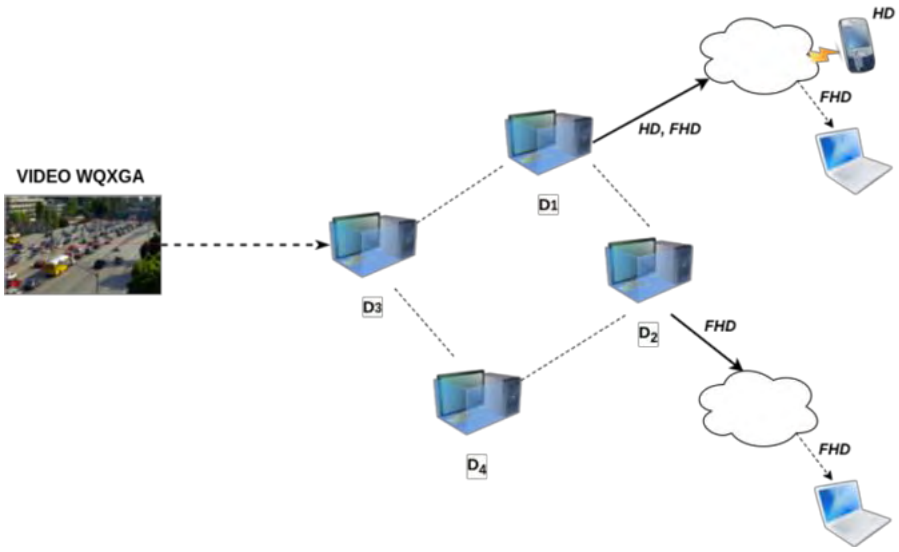


Figure 8-2 An example transcoding job with two corresponding tasks

transcoding (must be sent from D_2 to the region of D_1). The total network overhead in this case equals $7b/2$. On the other hand if all the transcoding tasks were assigned to D_4, the input sequence overhead would be b (D_3 to D_4 transfer) and the output overhead would be $(2+1)b/2$ for FullHD output and 2b/4 for HD, for a total overhead of 3b.

8.3. Scheduling algorithms

We consider a two level scheduling scheme. In the top level a global scheduler intercepts all job requests and assigns the corresponding tasks to data centers. At every data center a local scheduler is responsible to assign the tasks allocated by the global scheduler onto the available servers of the data center. Before proceeding with the description of local and global scheduling policies, we would like to mention that in practice a global scheduler isn't required to really intercept all job requests submitted by end users. Rather a policy can be implemented whereby the local scheduler of the data center a request arrives at, asks the global scheduler whether it should redirect the request or handle it itself, thus, realizing the global scheduler in an indirect manner. We refrain

from discussing such details, instead focusing on the questions of what information should be used by the global scheduler, what levels of cooperation (if any) with local schedulers are desirable and what selection policies to follow.

8.3.1. Local Scheduling

We experimented with HEFT-like local schedulers working as follows. Tasks are considered for scheduling one by one in order of their arrival time. In order to assign the task under examination to a server, all servers are examined with the aim of finding one that satisfies the processing time criterion. In case more than one such server exists, the more energy efficient one according to (8.1) is selected. For the case of live transcoding jobs the processing time criterion dictates finding the server such that q_{xyk} is maximized, while for non-live tasks, we consider a deadline that must be met. It is worth noting that as far as the global scheduler is concerned, local schedulers are mere black boxes that might even follow different strategies at different data centers. Thus, the particular selection (HEFT-based) made in the chapter is for simulation experiments purpose and doesn't restrict global scheduling choices.

8.3.2. Non Coordinated Global Scheduling

To be feasible, global schedulers must work without having full knowledge of data center details. For instance knowing server queues would require an unrealistically high monitoring effort. In this chapter we only consider the following information to be available at the global scheduling level: (a) the hop distance between any two data centers, i.e., h_{ij} values; (b) the total number of servers at each data center; (c) the average energy efficiency and processing capacity of the servers residing at each data center; (d) for each job J_x its characteristics and (e) for each data center, the tasks currently assigned to it that haven't finished yet. Information (a), (b) and (c), can be made available during a preprocessing step and doesn't change throughout system's operation time. Knowledge of (d) is straightforward requiring only the information available with each job request. Finally, (e) can be made available by having each local scheduler notifying the global scheduler upon a task's completion.

Using the above described information, the global scheduler estimates for each data center three values (e1, e2 and e3) each for the criteria of processing time, energy efficiency and network cost, measuring how "good" it would be to assign a particular task at the data center in question. For the first two criteria the following estimations are used:

$$e1(D_i) = \sum_{T_{xy}|\exists k: P_{xyk} B_{ik}=1} W_{xy} \Big/ \sum_{k=1}^{S} C_k B_{ik} \tag{8.9}$$

$$e2(D_i) = e1(D_i)\left(\frac{\sum_{k=1}^{S} B_{ik}}{\sum_{k=1}^{S} E_k B_{ik}}\right) \tag{8.10}$$

The third estimator e3, i.e., the one for network, is actually calculated in an exact manner using (8.4)-(8.8) since the necessary information to do so is available by (a) and (d).

The non-coordinating global scheduler works in the following manner. Upon a job's arrival, and for every possible task-data center assignment, it calculates estimators e1, e2 and e3. The smaller the value of an estimator is, the better the assignment for this particular performance metric. Various policies can then be acted upon. For instance, the scheduler might choose based on e1 only, e2, or e3, termed the resulting scheme accordingly as P (processing), E (energy), N (network). Another policy is to rate the best solution across all three metrics. For this reason data centers are sorted three times (for e1, e2 and e3) and the best Pareto optimal candidate is defined as the one with the minimum cumulative rank across all the three lists. Hence forth for sort, by Pareto optimal we will denote a solution with minimum cumulative ranking. We term this policy as PEN. Once a data center is selected, the global scheduler informs the relevant local scheduler upon its decision. The local scheduler can't deny a task assigned by the global one.

8.3.3. Coordinated Global Scheduling

Clearly, even if P and E heuristics indicate that a data center is a good choice for particular task allocation, it is possible that when the task is actually assigned there, the performance exhibited would be inferior to the one estimated. Therefore all policies (but N) might involve bad allocation decisions. To ameliorate the situation, here coordination strategies are discussed.

The scheme works as follows. First, e1 and e2 are estimated together with e3 which is accurately calculated. However, instead of deciding afterwards upon the datacenter to use for task processing, it asks the k most promising Pareto-wise local schedulers while making sure that the one with the best e1 value is included in this contact list. The local schedulers reply by sending the actual values that will incur (time and energy wise). Having gathered these replies, the coordination scheme decides the final assignment, using similar methods to the one described for the non-coordinating case, but having as candidates only the data centers conducted and using the actual values obtained from their answers instead of the estimations. The resulting schemes are termed accordingly as P-*k*, E-*k*, N-*k*, PEN-*k*.

8.4. Experiments

8.4.1. Simulation setup

Datacenters: To generate the network of datacenters we used the 30 regions where Azure was available as of November 2016 [15] assuming the existence of a single data center at each of them as shown in Figure 8-4. Due to absence of relevant connectivity information, we created the network as follows. First, we calculated for every Azure region pair, its geographic distance. Then at the resulting graph we run MST.

Servers: We performed simulation experiments on a Linux server with two 6-core Intel Xeon E5-2630 CPUs running at 2.3GHz. Since our server setting is not of generic use, we translated results into one of the Amazon EC2 instances to make our simulation setting more applicable. Specifically, we consider the C3 instances which are recommended for video coding. Comparing the processor passmark ratios between the CPU of our server and the one used in the C3 instances, i.e., Intel Xeon E5-2680 v2 (Ivy

	Resolution	Frames per second (fps)	Total frames	CTUs per frame
PeopleOnStreet	2560×1600	30	150	1000
Traffic	2560×1600	30	150	1000
BasketballDrive	1920×1080	50	500	510
BQTerrace	1920×1080	60	600	510
Cactus	1920×1080	50	500	510
Kimono	1920×1080	24	240	510
ParkScene	1920×1080	24	240	510

Table 8-2 Video Sequences

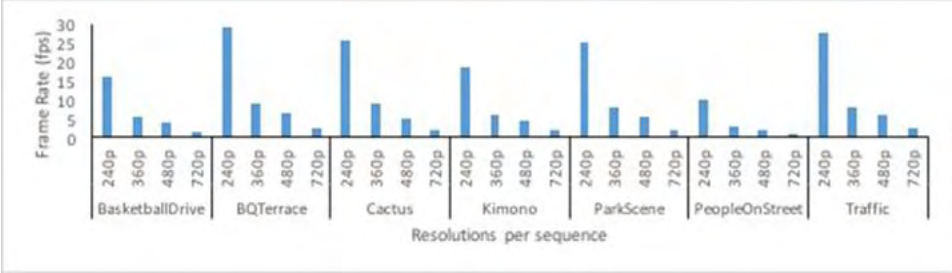


Figure 8-3 Frame rate coding times for common test video sequences in various resolutions

Figure 8-4 Azure regions

Bridge) it can be estimated that the instance c3.4xlarge will account for a speedup of 2.2x compared to our server. Thus, we estimated the coding times over C3 instances using the actual values obtained by our server factored by the aforementioned speedup.

Live casting dataset: To simulate broadcasting activity, we used the same dataset from Twitch as the one described in [13]. We kept the portion of the dataset representing one day activity (Jan. 6th, 2014). We then filtered it by deleting entries with broadcasts having no viewers and the broadcasts of resolution less than 220p. To keep the simulation time manageable we consider the following 5 resolutions: 240p, 360p, 480p, 720p and 1080p. In case a broadcast in the trace doesn't follow one of the previously mentioned resolutions, we cluster it to its closest matching. We assume that a broadcast must be transcoded to all the resolutions that are lower than the one it uses. Clearly, with this setting the maximum number of transcoding tasks incurred by a broadcast is 4, corresponding to a 1080p stream that must be downscaled to 720p, 480p, 360p and 240p. Upscaling, isn't considered in the experiments. For simulation purposes we assumed that all videos used 30 fps.

In the live casting scenario all arriving sequences were in H.264/AVC and should be transmitted in the same standard but at different resolution/rate. Since in the dataset of [13] no relevant information existed concerning transcoding times, we resorted to estimating them as follows. We used Class A and B common test (raw) video sequences with characteristics summarized in Table 8-2. Using ffmpeg we adjusted the resolution to the desired ones and encoded them using x264 with parameters suggested for PSNR tailored performance in [48]. Figure 8-3 plots the frame rate achieved by the encoder using one core. We then used the average value for its resolution case as a baseline estimator of the actual transcoding time. This baseline refers to the process of completely decoding the arriving sequence and re-encoding it from scratch, a strategy that is less than optimal as described in Section 2.2. To capture the case where a more efficient transcoder would be available we considered a speedup of 7x in the process similarly to the performance achieved in [60]. Finally we factored in the number of available cores in a C3 instance and the relevant speedup against our server.

File transcoding dataset: We consider the case where a large social media provider wants to translate a portion of its videos from H.264/AVC into HEVC as motivated in [90]. In lack of a suitable dataset we created one as follows. We selected one of the most popular video publishers in Facebook, NowThis, which accounted during a 6 month time period (July 2016 –Dec. 2016) for roughly 6 billion views according to [143]. We random sampled 20 videos that accounted for roughly 200M views and downloaded them in both available resolutions, 240p and 720p (2 of them were 1080p instead of 720p). We then transcoded them from H.264/AVC into HEVC using x265 with the PSNR tailored settings suggested in [48] and recorded the transcoding time for the two resolution levels. In the relevant experiment we assume a dataset size of multiple of these 40 files (20 videos in 2 resolutions) arrive in the system for processing.

8.5. Results

First we evaluated the performance of the algorithms in the live casting scenario. Broadcasters and viewers were randomly distributed over the available datacenters for the purpose of calculating network overhead. We measured the performance of the algorithms as the number of servers per datacenter increases from a baseline value obtained uniformly between 50 and 200 up to a 2x, 4x and 8x case (random between 400 and 1,600).

Figure 8-5 shows the percentage of live casting streams that were processed successfully, i.e., at their nominal rate of 30fps. As expected with increased processing power per data center the acceptance rate increases as well reaching 100% for all heuristics in the 8x case. Among single criterion options, the algorithm that favors processing time (P) over all other metrics achieves the best performance, followed by the network only metric (N) and the energy (E). It is worth noting, that the PEN algorithm that is based on identifying Pareto-optimality achieves comparable to P results in terms of accepted jobs. However, the merits of PEN are clearly shown in Figure 8-6 and Figure 8-7 that plot network overhead and energy consumption respectively. PEN achieves almost in par performance concerning the energy criterion against P while as far as network overhead is concerned, it accounts for a dramatic improvement over both P and E heuristics (about 4x less overhead). Compared to N that assigns jobs to the optimal

(network-wise) datacenters, PEN incurs roughly between 2x and 2.5x network cost (Figure 8-6), but accounts for better acceptance rates as shown in Figure 8-5. As far as the energy criterion is concerned, Figure 8-7 shows that especially when computational resources are limited (1x, 2x and 3x cases), E achieves the best performance with no clear winner for the second place.

Viewing the first results in retrospect, we can state that the merits of a scheduler operating with PEN are demonstrated by the fact that maximum job acceptance rate was achievable while cutting down significantly network overhead and without affecting at large energy consumption.

In the second experiment we evaluated the performance of the algorithms for the file transcoding Facebook scenario. We considered 20 servers per datacenter assigned for the transcoding service and plotted results as the number of arrived jobs increases. For each job we assumed a deadline equaling the sequence duration plus 10 minutes. Figure 8-8, Figure 8-9 and Figure 8-10, depict the percentage of jobs that meet their deadlines, the network and energy overhead respectively. Again PEN achieves top performance as far as processing time and deadlines are concerned (Figure 8-8) but at a smaller network overhead (Figure 8-9) compared to the other alternatives.

Last, we evaluated the performance of PEN-k that involves accurate estimations for the three metrics by coordinating with the k most promising datacenters. For the live casting scenario and for the 2x case, Figure 8-11 shows the expected improvement in terms of accepted jobs when using k=5, k=10 and k=15. Results show a small performance increase which might be deemed necessary in certain administration cases since it refers to acceptance rate. Nevertheless it should be noted that this improvement comes with a coordination overhead that increases linear to the number of datacenters involved (k), incurring an extra of $2k$ messages per task.

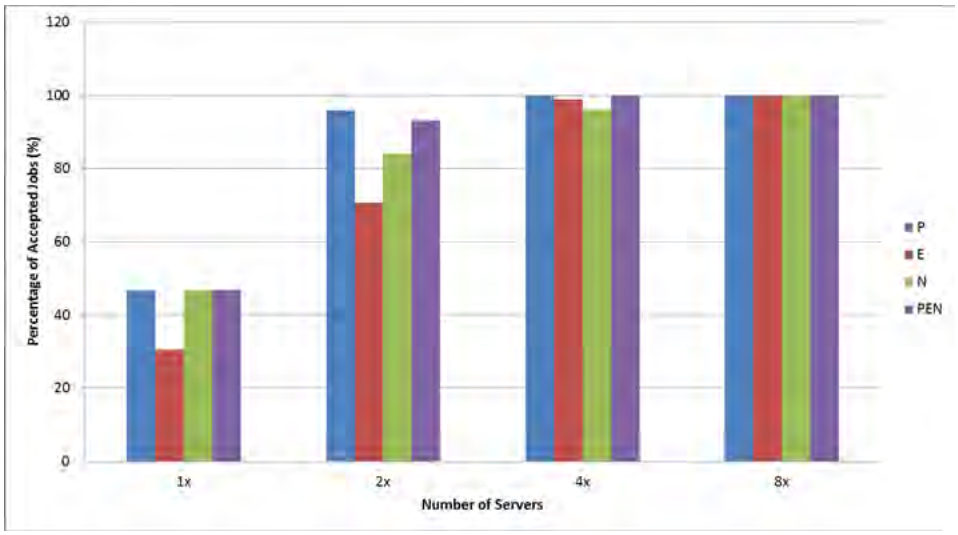


Figure 8-5 Percentage of accepted jobs as the number of servers per datacenter increases (live casting scenario).

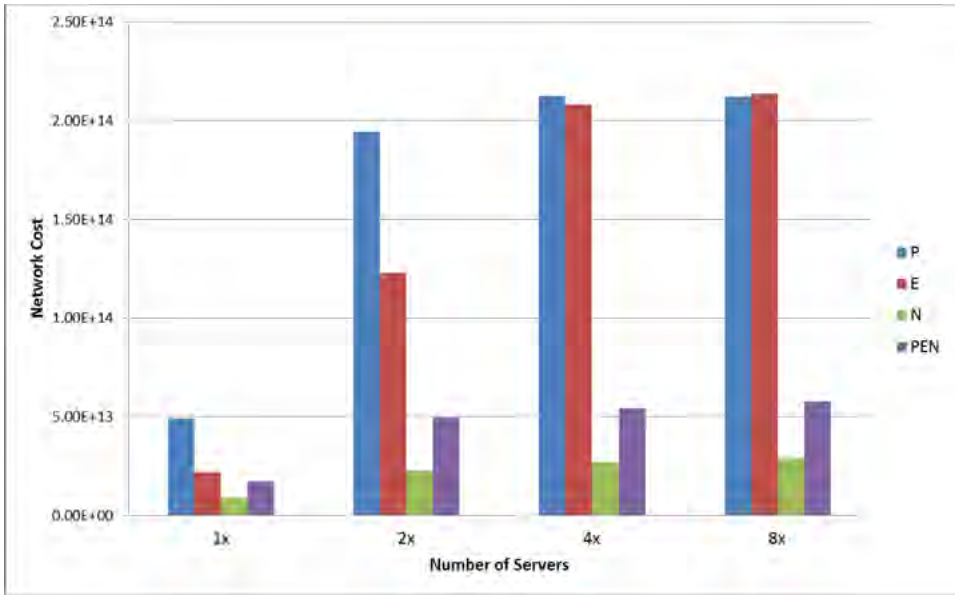


Figure 8-6 Network cost as the number of servers per datacenter increases (live casting scenario).

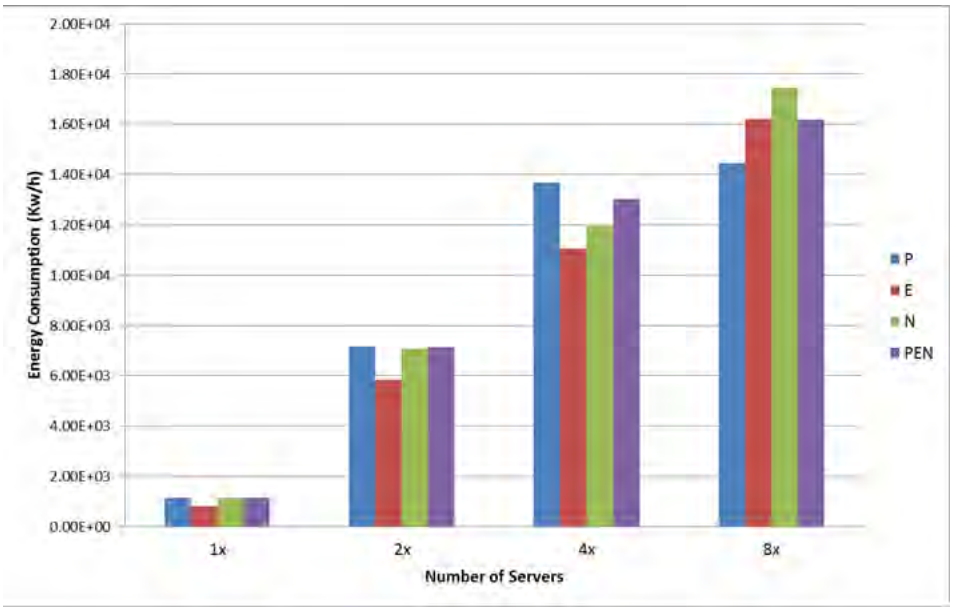


Figure 8-7 Energy consumption as the number of servers per datacenter increases (live casting scenario).

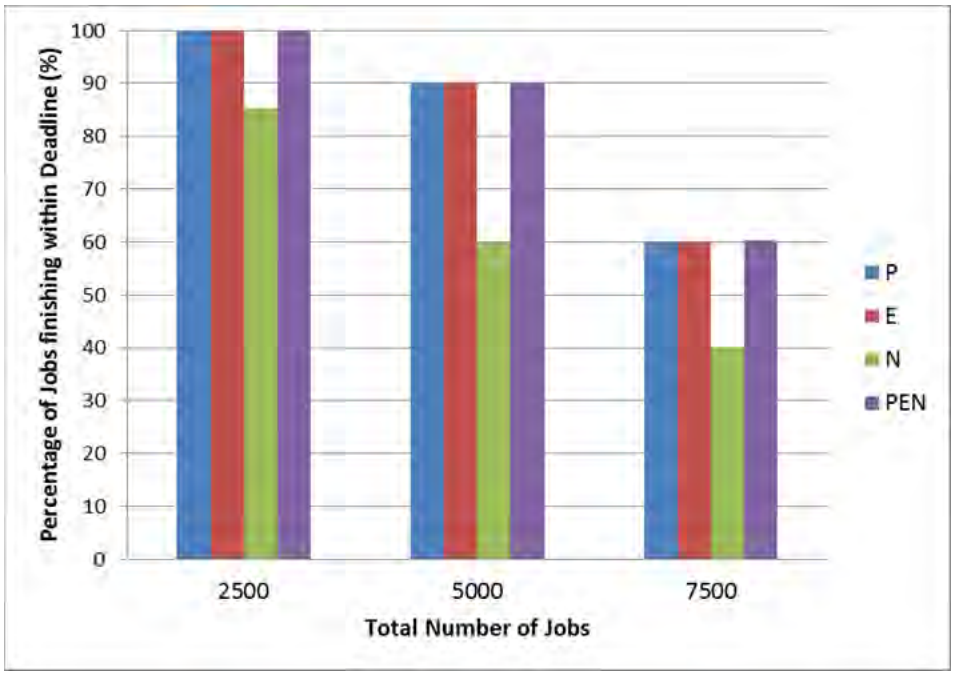


Figure 8-8 Percentage of jobs finishing within their deadlines as the number of servers per datacenter increases (Facebook scenario).

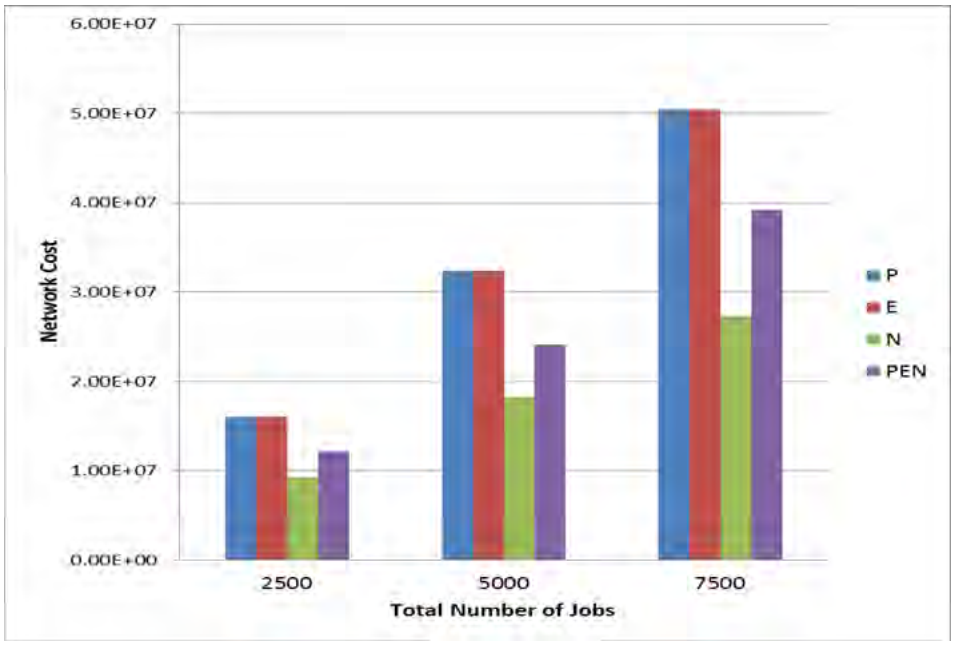


Figure 8-9 Network cost as the number of servers per datacenter increases (Facebook scenario).

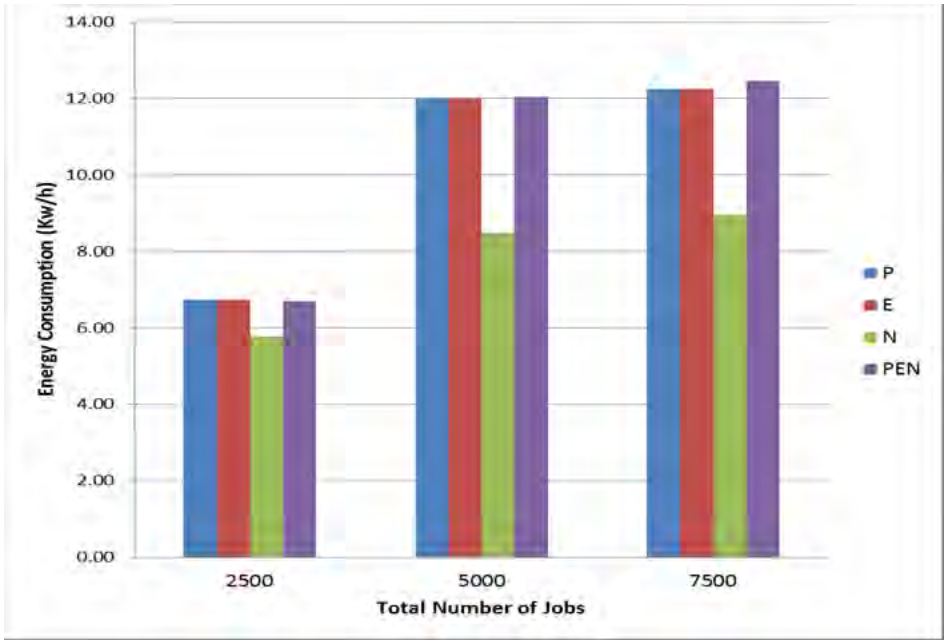


Figure 8-10 Energy consumption as the number of servers per datacenter increases (Facebook scenario).

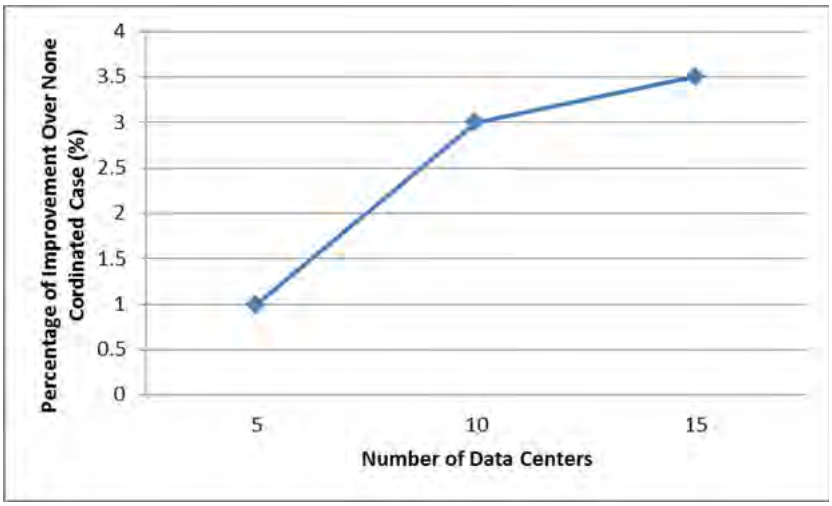


Figure 8-11 Improvement in accepted jobs (live casting scenario, PEN-k).

9. Combinatorial Optimization

9.1. Quadratic Assignment Problem (QAP)

Quadratic Assignment Problem (QAP) is one of the fundamental problems in the area of Combinatorial Optimization. It was originated [89] as a problem of allocating a finite set of facilities into a set of locations targeting the minimization of the overall allocation cost. Allocation cost is defined as the total sum, over all pairs, of the workflow between interconnected facilities multiplied by their distance respectively. Workflow costs and distances are known a priori while each facility can be assigned in exactly one location.

Given a set of positive integers $N = \{1, 2, \dots, n\}$ and a set of all permutations $S_n = \Phi: N \to N$ a formal mathematical definition of the QAP can be:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} * d_{\pi(i)\pi(j)} \tag{10.1}$$

over all permutations $\pi \in S_n$. Parameters $F = (f_{ij})$ and $D = (d_{ij})$ are nxn matrices denoting the required workflow between facilities *i,j* as well as the distance between them.

Considering the following Quadratic Assignment Problem consists of four facilities (f1, f2, f3, f4) and four locations (l1, l2, l3, l4). Table 9-1 and Table 9-2 denote the workflow and the distance respectively.

0	22	53	53
22	0	40	62
53	40	0	55
53	62	55	0

Table 9-1 Facilities Workflow F

0	22	53	53
22	0	40	62
53	40	0	55
53	62	55	0

Table 9-2 Facilities Distance D

A possible facilities placement is μ={4,3,1,2} indicating that facility f4 is placed into location l1, facility f_3 is placed into location l_2 and so on. The overall assignment cost, as show in Figure 9-1 is $f_{12} * d_{l_3 l_4} + f_{14} * d_{l_3 l_1} + f_{24} * d_{l_4 l_1} + f_{34} * d_{l_2 l_1} = 412$. However, placement μ is a local optimal. Figure 9-2 shows the global optimal μ={3,4,1,2} resulting a total assignment cost equal to 395, while the individual cost between interconnected facilities is presented over the edges.

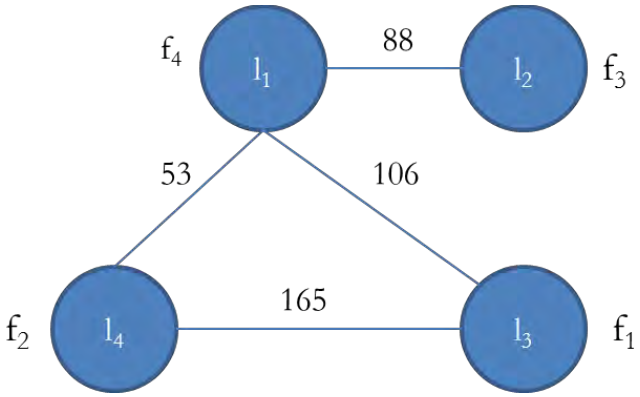


Figure 9-1 Local optimal assignment μ={4,3,1,2}

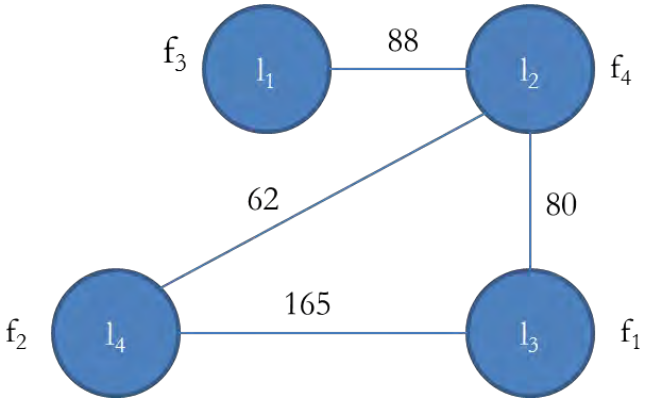


Figure 9-2 Global optimal assignment μ={3,4,1,2}

Aforementioned example practicality is two folded. On the one hand it clarifies the usefulness on applying QAP into real world problems; on the other hand it reveals the difficulty on finding an optimal solution even for a small case of four facilities. Meanwhile, for large number of test cases $n>30$ solving QAP in a reasonable time seems impossible [120]. Methods used on solving QAP can be classified into two major categories [104]. Approaches in the first category comes up with optimal solution using Branch and Bound strategies [111], [69] and Dynamic Programming [36] while, in the second category heuristic methods used where optimal solution is not guaranteed [26], [119]. QAP was also widely used by consolidating metaheuristic techniques such as simulated annealing [116], Genetic algorithms [138] and swarm intelligence [107].

In the years to come, this optimization model will prove particularly important on solving complex problems that will belong to different research areas. Steinberg [131] apply QAP for the problem of placing interconnecting computer components into a backboard so as to minimize the total amount of wire needed which affects the timing and the overall performance of the circuits. Particular emphasis was given to solving problems that affects the everyday life such as the hospital layout problem [56], [108] and the arrangement of campus buildings [93]. Besides that QAP used in the field of architecture [93] and sports [55] giving optimal designs of a structure's layout subject to certain conditions as well as maximizing the overall risk taken by a player. As far as the hospital layout problem is concerned QAP was proposed for the minimization of the

total annual distance covered by patients formed through the allocation of 17 departments into 17 clinics. In the same manner the weekly distance covered among departments in a campus was decreased thus bottlenecks have been rooted out [52]. In light of parallel and distributed systems [20], [41] QAP used to increase solvable capability in complex benchmarks [111], while in a scheduling problem [69] integrated with dynamic programming led to a reduction in production cost.

9.2. File Allocation Problem (FAP)

File Allocation Problem (FAP) is a well-known combinatorial optimization problem concerning the optimal distribution of files among different computers in a distributing environment. Different parameters are considered in the FAP for instance the number of file copies to be stored and the copies locations both with respecting the minimization of operation costs like file storage, query, update and communication [54]. FAP initially formulated via integer programing or via linear programing ignoring integer restrictions in terms of different objective functions and constrains [31], [39]. Heuristics like efficient Branch and Bound and greedy algorithms revealed in the near future [66], [148] some of them tackle the FAP assuming tractable assumptions and scenarios [32].

9.3. Capacitated Plant Location Problem

Capacitated Plant Location Problem (CPLP) falls in the area of combinatorial optimization since the middle of the previous century [130]. It concerns the optimal distribution of plants into potential locations over a distributed production network targeting the minimization of two significant operation costs. Supplying commodities to clients through the network yields a noticeable operation cost while opening, maintain and running a plant in a specific location is a key issue in the CPLP. Aforementioned operation costs differ on various locations and plant sizes; meanwhile each plant has an upper bound on the amount of demand it can service and customer demands must be fulfilled. Solving CPLP is usually accomplished in two phases. In the first one a subset of plants is considered to be open while in the second phase customers are assigned into opened plants. Solving CPLP poses interesting dilemmas. Exact algorithms don't scale well for large number of inputs while heuristics algorithms may not converge to global optimal. ADD, DROP [94], [65] procedures and generalizations of them [78] belongs in

the area of greedy heuristics where once a decision is made it doesn't change while interchange heuristics [57], [134], [140] attempt to alter decisions in an overall minimization manner. Applications of CPLP are not restricted into industry related problems. For instance in routing decision [59] CPLP used for the minimization of the total delivery cost which occurs by chauffeuring demand in different locations. Lock box location problem [109] is another example where CPLP was successfully leading banks to manage efficient their profit. CPLP is also used for the installation of databases [58]. Accessing remote databases, especially with the continuous growth in the wide area networks raises significant communication and operation costs thereby assigning copies of important databases into different locations potentially minimize the aforementioned costs.

9.4. Web proxy placement

Effective use of network bandwidth is a significant concern the last decades especially with an ever increasing demands on users using the internet and on data that being saved and transferred all over the world. Satisfying users demand while preserving bandwidth usage and traffic congestion as low as possible act as a cutting edge for both researchers and internet server providers (ISPs) Web cashing is a technology used for better bandwidth allocation in conjunction with response time minimization for the common good of users and companies requirements. Essentially, web cache technology deals with the creation of temporary data copies usually contain html files and images which may be needed frequently over the internet. In this way data is transferred to clients via cache instead of transferring data from actual locations, as a result data transmission to end users is done faster and servers load – bandwidth allocation is decreased. Cashing data close to clients is considered an effective solution for bandwidth management problems and for server scalability [149]. Installing caching schemes near clients is usually tackled via establishing proxy on firewall machines as seen in Figure 9-3. In more generic models as presented in Figure 9-4, upon a client request a proxy that doesn't have the required information communicates with others affiliated proxies. In doing so, in an efficient manner a proxy placement methodology must be used. In [100] authors use dynamic programming to deal with optimal placement of multiple proxies among different locations (sites) based on a given traffic. Cache replacement techniques also used for the

minimization of different objectives. Namely, traditional techniques like selecting the data which was requested the least recently, key based approaches such as LRU-MIN and LRU-Threshold [3] and cost based policies where potential costs are assigned to the metrics we want to optimize. Efficient placement affects the overall web performance as it reduces latency access, the bandwidth consumption and the workload of remote servers especially in the case where a remote server is unreachable.

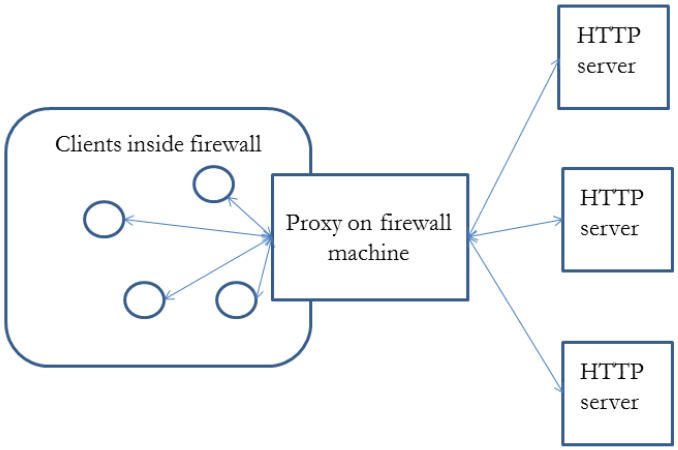


Figure 9-3 Installing Proxy on a firewall machine

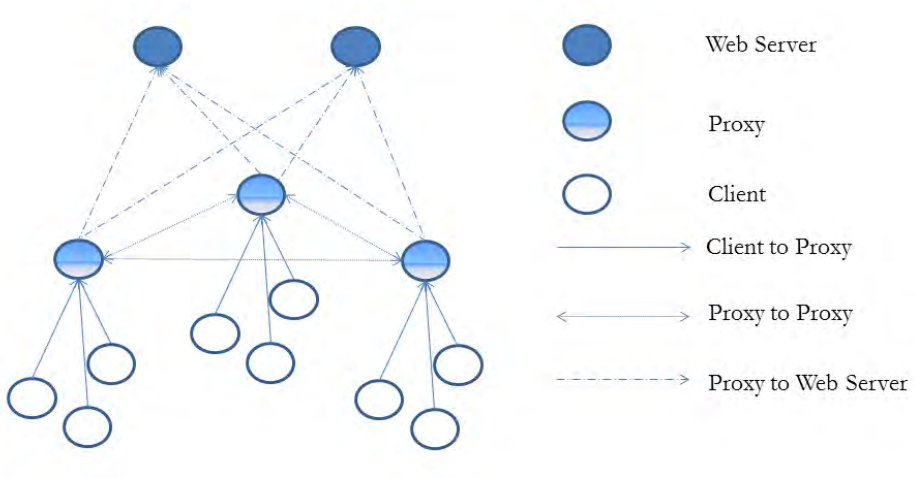


Figure 9-4 A generic cashing scheme.

10. Concussions and Future Research

In this chapter we summarize the contributions of this thesis and discuss the important directions of future work. Main conclusions emerging from this work are the following:

- Tackling IC placement as a scheduling problem is feasible. Critical path delay optimization can be achieved by placing potentially slow components, being individual cells or paths, close to power sources.

- Performance of a classic legalization algorithm, Tetris, can be augmented, while retaining its fast execution time. It turns out that drastic performance improvement in HPWL (which directly translates to power improvement) and displacement terms is achievable by AC4-LR, AC4-RR10% and CC8-LR. The first two heuristics also improve execution time, while the last heuristic achieves placement quality that puts it almost in par with a more complex legalization scheme (Abacus) for NTUplace3 input but at a two orders of magnitude less runtime. Thus, whenever fast legalization is needed, the heuristics presented in the thesis offer better alternatives compared to both Tetris and Abacus.

- High execution time of state of the art legalizers can be decreased via parallelism (coarse grained parallelization). Instead of following a one pass approach that needed extensive synchronization overhead, we followed a two pass approach, consisting of a first parallel, lock-free step followed by a sequential one. The same methodology with minimal or no changes can be applied on most legalization algorithms of the literature given that they don't form their own partitioning. Its particular implementation on Abacus, called Domocus, proved to augment the time performance of the sequential scheme, without seriously affecting solution quality, even improving it in certain cases. Furthermore, Cloud environment fits IC placement requirements.

- Video coding and transcoding are computationally demanding, performing a portion of these tasks at the network edges decrease both the workload and network traffic towards the data centers of media providers. Also, a small quality degradation of the output video streams leads to an increasing number of jobs assigned to the edge datacenter.

- Scheduling transcoding jobs over the cloud poses interesting dilemmas. Different strategies can be applied including light and strong cooperation mechanisms and multiple objective functions like processing time, energy consumption and network cost. Results show that a heuristic that tackles all three optimization targets using a Pareto-optimal approach can lead to significant network savings with marginal effects on task completion time.

As part of our future work we plan to study and implement more sophisticated parallel heuristics which can prone sufficiently the search space without significant losses and incorporate multi-objective optimizers that take into account various objective functions Moreover, we will focus on creating cloud application services (SaaS) which will use the web to deliver IC placement and video transcoding applications to clients. These services will serve the needs of both researchers and academia.

BIBLIOGRAPHY

[1]. Aaron, A., Li, Z., Manohara, M., Cock, J. D., & Ronca, D. Per-Title Encode Optimization. The Netflix Tech Blog (on-line).

[2]. Ababei, Cristinel, et al. "Multi-objective circuit partitioning for cutsize and path-based delay minimization." Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design. ACM, 2002.

[3]. Abrams, Marc, et al. "Caching proxies: Limitations and potentials." (1995).

[4]. Agarwal, D., & Jain, S. (2014). Efficient optimal algorithm of task scheduling in cloud computing environment. arXiv preprint arXiv:1404.2076.

[5]. Agnihotri, Ameya, et al. "Fractional cut: Improved recursive bisection placement." Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design. IEEE Computer Society, 2003.

[6]. Agnihotri, Ameya R., Satoshi Ono, and Patrick H. Madden. "Recursive bisection placement: Feng Shui 5.0 implementation details." Proceedings of the 2005 international symposium on Physical design. ACM, 2005.

[7]. Ahmad, I., Wei, X., Sun, Y., & Zhang, Y. Q. (2005). Video transcoding: an overview of various techniques and research issues. IEEE Transactions on multimedia, 7(5), 793-804.

[8]. Ahn, Y. J., Hwang, T. J., Sim, D. G., & Han, W. J. (2014). Implementation of fast HEVC encoder based on SIMD and data-level parallelism. EURASIP Journal on Image and Video Processing, 2014(1), 16.

[9]. Alliance for Open Media. Available at: http://aomedia.org

[10]. Alpert, Charles Jay, et al. "Porosity aware buffered steiner tree construction." U.S. Patent No. 7,065,730. 20 Jun. 2006.

[11]. Amazon Elastic Transcoder. Available at: https://aws.amazon.com/elastictranscoder/

[12]. Amazon Web Services. (2017, Jan. 30). Amazon EC2 Instance Types [Online]. Available: https://aws.amazon.com/ec2/instance-types/

BIBLIOGRAPHY

[13]. Aparicio-Pardo, R., Pires, K., Blanc, A., & Simon, G. (2015, March). Transcoding live adaptive video streams at a massive scale in the cloud. In Proceedings of the 6th ACM Multimedia Systems Conference (pp. 49-60). ACM.

[14]. Ashraf, A., Jokhio, F., Deneke, T., Lafond, S., Porres, I., & Lilius, J. (2013, May). Stream-based admission control and scheduling for video transcoding in cloud computing. In Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on (pp. 482-489). IEEE.

[15]. Azure Regions. Available at: https://azure.microsoft.com/en-us/regions/?cdn=disable

[16]. Beck, M. T., Feld, S., Fichtner, A., Linnhoff-Popien, C., & Schimper, T. (2015, February). ME-VoLTE: Network functions for energy-efficient video transcoding at the mobile edge. In Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on (pp. 38-44). IEEE.

[17]. Beloglazov, A., & Buyya, R. (2012). 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.

[18]. Beloglazov, A., Buyya, R., Lee, Y. C., & Zomaya, A. (2011). A taxonomy and survey of energy-efficient data centers and cloud computing systems. Advances in computers, 82(2), 47-111.

[19]. Bilal, K., Fayyaz, A., Khan, S. U., & Usman, S. (2015). Power-aware resource allocation in computer clusters using dynamic threshold voltage scaling and dynamic voltage scaling: comparison and analysis. Cluster Computing, 18(2), 865-888.

[20]. Bokhari, Shahid H. Assignment problems in parallel and distributed computing. Vol. 32. Springer Science & Business Media, 2012.

[21]. Bossen, Frank. "Common test conditions and software reference configurations." Joint Collaborative Team on Video Coding (JCT-VC), JCTVC-F900 (2011).

[22]. Brenner, Ulrich. "VLSI legalization with minimum perturbation by iterative augmentation." Proceedings of the Conference on Design, Automation and Test in Europe. EDA Consortium, 2012.

BIBLIOGRAPHY

[23]. Brenner, Ulrich, and Jens Vygen. "Faster optimal single-row placement with fixed ordering." Proceedings of the conference on Design, automation and test in Europe. ACM, 2000.

[24]. Brenner, Ulrich, and Jens Vygen. "Legalizing a placement with minimum total movement." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 23.12 (2004): 1597-1613.

[25]. Brenner, Ulrich, Markus Struzyna, and Jens Vygen. "BonnPlace: Placement of leading-edge chips by advanced combinatorial algorithms." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27.9 (2008): 1607-1620.

[26]. Buffa, Elwood Spencer, Gordon C. Armour, and Thomas E. Vollmann. Allocating facilities with CRAFT. Boston, MA: Harvard University, 1964.

[27]. Caldwell, Andrew E., Andrew B. Kahng, and Igor L. Markov. "Optimal partitioners and end-case placers for standard-cell layout." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 19.11 (2000): 1304-1313.

[28]. Casavant, T. L., & Kuhl, J. G. (1988). A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Transactions on software engineering, 14(2), 141-154.

[29]. Chan, Tony F., et al. "mPL6: enhanced multilevel mixed-size placement." Proceedings of the 2006 international symposium on Physical design. ACM, 2006.

[30]. Chandio, A. A., Bilal, K., Tziritas, N., Yu, Z., Jiang, Q., Khan, S. U., & Xu, C. Z. (2014). A comparative study on resource allocation and energy efficient job scheduling strategies in large-scale parallel computing systems. Cluster computing, 17(4), 1349-1367.

[31]. Chandy, K. Mani, and J. E. Hewes. "File allocation in distributed systems." Proceedings of the 1976 ACM SIGMETRICS conference on Computer performance modeling measurement and evaluation. ACM, 1976.

[32]. Chen, Peter PS. "Optimal file allocation in multi-level storage systems." Proceedings of the June 4-8, 1973, national computer conference and exposition. ACM, 1973.

BIBLIOGRAPHY

[33]. Chen, Tung-Chieh, et al. "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27.7 (2008): 1228-1240.

[34]. Cheon, Yongseok, et al. "Power-aware placement." Proceedings of the 42nd annual Design Automation Conference. ACM, 2005.

[35]. Chi, C. C., Alvarez-Mesa, M., Juurlink, B., Clare, G., Henry, F., Pateux, S., & Schierl, T. (2012). Parallel scalability and efficiency of HEVC parallelization approaches. IEEE Transactions on Circuits and Systems for Video Technology, 22(12), 1827-1838.

[36]. Christofides, Nicos, and Enrique Benavent. "An exact algorithm for the quadratic assignment problem on a tree." Operations Research 37.5 (1989): 760-768.

[37]. Cho, Minsik, et al. "History-based VLSI legalization using network flow." Proceedings of the 47th Design Automation Conference. ACM, 2010.

[38]. Chou, Sheng, and Tsung-Yi Ho. "OAL: An obstacle-aware legalization in standard cell placement with displacement minimization." SOC Conference, 2009. SOCC 2009. IEEE International. IEEE, 2009.

[39]. Chu, Wesley W. "Optimal file allocation in a multiple computer system." IEEE Transactions on Computers 100.10 (1969): 885-889.

[40]. Cisco, Cisco Visual Networking Index. "Global Mobile Data Traffic Forecast Update, 2015–2020 White Paper, 2016."

[41]. Clausen, Jens, and Michael Perregaard. "Solving large quadratic assignment problems in parallel." Computational Optimization and Applications 8.2 (1997): 111-127.

[42]. Cong, Jason, and Min Xie. "A robust detailed placement for mixed-size IC designs." Design Automation, 2006. Asia and South Pacific Conference on. IEEE, 2006.

[43]. Dadaliaris, A. N., Nerantzaki, E., Koziri, M. G., Oikonomou, P., Loukopoulos, T., & Stamoulis, G. I. (2016, November). Performance Evaluation of Tetris-based Legalization Heuristics. In Proceedings of the 20th Pan-Hellenic Conference on Informatics (p. 60). ACM.

[44]. Dadaliaris, A. N., Oikonomou, P., Koziri, M. G., Nerantzaki, E., Hatzaras, Y., Garyfallou, D., ... & Stamoulis, G. I. (2017). Heuristics to Augment the Performance of Tetris Legalization: Making a Fast but Inferior Method Competitive. Journal of Low Power Electronics, 13(2), 220-230.

[45]. Dakshayini, D. M., & Guruprasad, D. H. (2011). An optimal model for priority based service scheduling policy for cloud computing environment. International Journal of Computer Applications, 32(9), 23-29.

[46]. Darav, Nima Karimpour, et al. "High performance global placement and legalization accounting for fence regions." Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. IEEE Press, 2015.

[47]. Dayarathna, Miyuru, Yonggang Wen, and Rui Fan. "Data center energy consumption modeling: A survey." IEEE Communications Surveys & Tutorials 18.1 (2016): 732-794.

[48]. De Cock, J., Mavlankar, A., Moorthy, A., & Aaron, A. (2016, September). A large-scale video codec comparison of x264, x265 and libvpx for practical VOD applications. In SPIE Optical Engineering+ Applications (pp. 997116-997116). International Society for Optics and Photonics.

[49]. De la Torre, E., Rodriguez-Sanchez, R., & Martínez, J. L. (2015). Fast video transcoding from HEVC to VP9. IEEE Transactions on Consumer Electronics, 61(3), 336-343.

[50]. Delavar, A. G., Javanmard, M., Shabestari, M. B., & Talebi, M. K. (2012). RSDC (reliable scheduling distributed in cloud computing). International Journal of Computer Science, Engineering and Applications, 2(3), 1.

[51]. Delimitrou, C., & Kozyrakis, C. (2013). The netflix challenge: Datacenter edition. IEEE Computer Architecture Letters, 12(1), 29-32.

[52]. Dickey, J. W., and J. W. Hopkins. "Campus building arrangement using TOPAZ." Transportation Research 6.1 (1972): 59-68.

[53]. Doll, Konrad, Frank M. Johannes, and Kurt J. Antreich. "Iterative placement improvement by network flow methods." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 13.10 (1994): 1189-1200.

[54]. Dowdy, Lawrence W., and Derrell V. Foster. "Comparative models of the file assignment problem." ACM Computing Surveys (CSUR) 14.2 (1982): 287-313.

[55]. Eiselt, Horst A., and Gilbert Laporte. "A combinatorial optimization problem arising in dartboard design." Journal of the Operational Research Society 42.2 (1991): 113-118.

[56]. Elshafei, Alwalid N. "Hospital layout as a quadratic assignment problem." Journal of the Operational Research Society 28.1 (1977): 167-179.

[57]. Feldman, E. F. A. T. L., F. A. Lehrer, and T. L. Ray. "Warehouse location under continuous economies of scale." Management Science 12.9 (1966): 670-684.

[58]. Fisher, Marshall L., and Dorit S. Hochbaum. "Database location in computer networks." Journal of the ACM (JACM) 27.4 (1980): 718-735.

[59]. Fisher, Marshall L., and Ramchandran Jaikumar. "A generalized assignment heuristic for vehicle routing." Networks 11.2 (1981): 109-124.

[60]. Franche, Jean-François, and Stéphane Coulombe. "Fast H. 264 to HEVC transcoder based on post-order traversal of quadtree structure." Image Processing (ICIP), 2015 IEEE International Conference on. IEEE, 2015.

[61]. Future Video Coding (JVET). Available at: https://jvet.hhi.fraunhofer.de

[62]. Gao, G., Zhang, W., Wen, Y., Wang, Z., & Zhu, W. (2015). Towards cost-efficient video transcoding in media cloud: Insights learned from user viewing patterns. IEEE Transactions on Multimedia, 17(8), 1286-1296.

[63]. Garey, M. R., & Johnson, D. S. (2002). Computers and intractability (Vol. 29). New York: wh freeman.

[64]. Geoffrion, A., and R. Me Bride. "Lagrangean relaxation applied to capacitated facility location problems." AIIE transactions 10.1 (1978): 40-47.

[65]. Geoffrion, Arthur M., and Glenn W. Graves. "Scheduling parallel production lines with changeover costs: Practical application of a quadratic assignment/LP approach." Operations Research 24.4 (1976): 595-610.

[66]. Ghosh, Deb, Ishwar Murthy, and Allen Moffett. "File allocation problem: Comparison of models with worst case and average communication delays." Operations Research 40.6 (1992): 1074-1085.

[67]. Gilmore, P. C., & Gomory, R. E. (1965). Multistage cutting stock problems of two and more dimensions. Operations research, 13(1), 94-120.

BIBLIOGRAPHY

[68]. Google Cloud Engine. Available: https://cloud.google.com/compute/

[69]. Graves, Glenn William, and Andrew B. Whinston. "An algorithm for the quadratic assignment problem." Management Science 16.7 (1970): 453-471.

[70]. Grois, D., Marpe, D., Mulayoff, A., Itzhaky, B., & Hadar, O. (2013, December). Performance comparison of h. 265/mpeg-hevc, vp9, and h. 264/mpeg-avc encoders. In Picture Coding Symposium (PCS), 2013 (pp. 394-397). IEEE.

[71]. Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems, 29(7), 1645-1660.

[72]. Hameed, A., Khoshkbarforoushha, A., Ranjan, R., Jayaraman, P. P., Kolodziej, J., Balaji, P., ... & Khan, S. U. (2016). A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. Computing, 98(7), 751-774.

[73]. Hill, Dwight. "Method and system for high speed detailed placement of cells within an integrated circuit design." U.S. Patent No. 6,370,673. 9 Apr. 2002.

[74]. Ho, Tsung-Yi, and Sheng-Hung Liu. "Fast legalization for standard cell placement with simultaneous wirelength and displacement minimization." VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP. IEEE, 2010.

[75]. Hsu, Meng-Kai, and Yao-Wen Chang. "Unified analytical global placement for large-scale mixed-size circuit designs." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 31.9 (2012): 1366-1378.

[76]. Hsu, Meng-Kai, Yao-Wen Chang, and Valeriy Balabanov. "TSV-aware analytical placement for 3D IC designs." Proceedings of the 48th Design Automation Conference. ACM, 2011.

[77]. Hur, Sung Woo, and John Lillis. "Mongrel: hybrid techniques for standard cell placement." Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design. IEEE Press, 2000.

[78]. Jacobsen, Soren Kruse. "Heuristics for the capacitated plant location model." European Journal of Operational Research 12.3 (1983): 253-261.

BIBLIOGRAPHY

[79]. Ji, Wen, Zhu Li, and Yiqiang Chen. "Content-aware utility-fair video streaming in wireless broadcasting networks." Image Processing (ICIP), 2011 18th IEEE International Conference on. IEEE, 2011.

[80]. Ji, Wen, Zhu Li, and Yiqiang Chen. "Joint source-channel coding and optimization for layered video broadcasting to heterogeneous devices." IEEE Transactions on Multimedia 14.2 (2012): 443-455.

[81]. Ibrahim, S., Phan, T. D., Carpen-Amarie, A., Chihoub, H. E., Moise, D., & Antoniu, G. (2016). Governing energy consumption in hadoop through cpu frequency scaling: An analysis. Future Generation Computer Systems, 54, 219-232.

[82]. Ioannidis, S. K., Ntioudis, D., Antoniadis, C., Dadaliaris, A. N., Tsompanopoulou, P., Evmorfopoulos, N. E., & Stamoulis, G. I. (2014). Optimization of an Integrated Circuit Placement Algorithm in a Parallel Environment. CSCESM2014, 179-192.

[83]. ISPD98 benchmark circuits. Available at: http://vlsicad.ucsd.edu/UCLAWeb/cheese/ispd98.html

[84]. Kahng, Andrew B., and Qinke Wang. "A faster implementation of APlace." Proceedings of the 2006 international symposium on Physical design. ACM, 2006.

[85]. Kahng, Andrew B., Igor L. Markov, and Sherief Reda. "On legalization of row-based placements." Proceedings of the 14th ACM Great Lakes symposium on VLSI. ACM, 2004.

[86]. Kim, Myung-Chul, et al. "A SimPLR method for routability-driven placement." Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on. IEEE, 2011.

[87]. Khaitan, Siddhartha Kumar, and James D. McCalley. "Design techniques and applications of cyberphysical systems: A survey." IEEE Systems Journal 9.2 (2015): 350-365.

[88]. Kleinhans, Jürgen M., et al. "GORDIAN: VLSI placement by quadratic programming and slicing optimization." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 10.3 (1991): 356-365.

[89]. Koopmans, Tjalling C., and Martin Beckmann. "Assignment problems and the location of economic activities." Econometrica: journal of the Econometric Society (1957): 53-76.

[90]. Koziri, M., Papadopoulos, P., Tziritas, N., Dadaliaris, A. N., Loukopoulos, T., & Khan, S. U. (2016, September). Slice-based parallelization in HEVC encoding: Realizing the potential through efficient load balancing. In Multimedia Signal Processing (MMSP), 2016 IEEE 18th International Workshop on (pp. 1-6). IEEE.

[91]. Koziri, M., Papadopoulos, P., Tziritas, N., Dadaliaris, A. N., Loukopoulos, T., & Stamoulis, G. I. (2016, October). A framework for scheduling the encoding of multiple smart user videos. In Semantic and Social Media Adaptation and Personalization (SMAP), 2016 11th International Workshop on (pp. 29-34). IEEE.

[92]. Koziri, M., Zacharis, D., Katsavounidis, I., & Bellas, N. (2011). Implementation of the AVS video decoder on a heterogeneous dual-core SIMD processor. IEEE Transactions on Consumer Electronics, 57(2).

[93]. Krarup, Jakob, and Peter Mark Pruzan. "Computer-aided layout design." Mathematical programming in use (1978): 75-94.

[94]. Kuehn, Alfred A., and Michael J. Hamburger. "A heuristic program for locating warehouses." Management science 9.4 (1963): 643-666.

[95]. Kwok, Yu-Kwong, and Ishfaq Ahmad. "Static scheduling algorithms for allocating directed task graphs to multiprocessors." ACM Computing Surveys (CSUR) 31.4 (1999): 406-471.

[96]. Lang, W., & Patel, J. M. (2010). Energy management for mapreduce clusters. Proceedings of the VLDB Endowment, 3(1-2), 129-139.

[97]. LD, D. B., & Krishna, P. V. (2013). Honey bee behavior inspired load balancing of tasks in cloud computing environments. Applied Soft Computing, 13(5), 2292-2303.

[98]. Lee, Yu-Min, Tsung-You Wu, and Po-Yi Chiang. "A hierarchical bin-based legalizer for standard-cell designs with minimal disturbance." Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific. IEEE, 2010.

BIBLIOGRAPHY

[99]. Leverich, J., & Kozyrakis, C. (2010). On the energy (in) efficiency of hadoop clusters. ACM SIGOPS Operating Systems Review, 44(1), 61-65.

[100]. Li, Bo, et al. "On the optimal placement of web proxies in the internet." INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Vol. 3. IEEE, 1999.

[101]. Lin, C., & Lu, S. (2011, July). Scheduling scientific workflows elastically for cloud computing. In Cloud Computing (CLOUD), 2011 IEEE International Conference on (pp. 746-747). IEEE.

[102]. Lin, S., Zhang, X., Yu, Q., Qi, H., & Ma, S. (2013, May). Parallelizing video transcoding with load balancing on cloud computing. In Circuits and Systems (ISCAS), 2013 IEEE International Symposium on (pp. 2864-2867). IEEE.

[103]. Lu, Jingwei, et al. "ePlace: Electrostatics-Based Placement Using Fast Fourier Transform and Nesterov's Method." ACM Transactions on Design Automation of Electronic Systems (TODAES) 20.2 (2015): 17.

[104]. Loiola, Eliane Maria, et al. "A survey for the quadratic assignment problem." European journal of operational research 176.2 (2007): 657-690.

[105]. Malik, S. U. R., Khan, S. U., Ewen, S. J., Tziritas, N., Kolodziej, J., Zomaya, A. Y., ... & Malluhi, Q. M. (2016). Performance analysis of data intensive cloud systems based on data management and replication: a survey. Distributed and Parallel Databases, 34(2), 179-215.

[106]. Markov, Igor L., Jin Hu, and Myung-Chul Kim. "Progress and challenges in VLSI placement research." Proceedings of the International Conference on Computer-Aided Design. ACM, 2012.

[107]. Maniezzo, Vittorio. "Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem." INFORMS journal on computing 11.4 (1999): 358-369.

[108]. Motaghi, Monika, et al. "Optimization of Hospital Layout through the Application of Heuristic Technique (Diamond Algorithm) in Shafa Hospital (2009)." International Journal of Management and Business Research 1.3 (2011): 133-138.

[109]. Nauss, Robert M., and Robert E. Markland. "Theory and application of an optimizing procedure for lock box location analysis." Management Science 27.8 (1981): 855-865.

[110]. Naylor, William C., Ross Donelly, and Lu Sha. "Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer." U.S. Patent No. 6,301,693. 9 Oct. 2001.

[111]. Nugent, Christopher E., Thomas E. Vollmann, and John Ruml. "An experimental comparison of techniques for the assignment of facilities to locations." Operations research 16.1 (1968): 150-173.

[112]. Oikonomou, P., Koziri, M. G., Dadaliaris, A. N., Loukopoulos, T., & Stamoulis, G. I. (2017, May). Domocus: Lock free parallel legalization in standard cell placement. In Modern Circuits and Systems Technologies (MOCAST), 2017 6th International Conference on (pp. 1-4). IEEE.

[113]. Oikonomou. P., Koziri, M. G.i, Tziritas, N., Loukopoulos, T., and Cheng-Zhong, X. Scheduling Heuristics for Live Video Transcoding on Cloud Edges. zTE Communications, DOI: 10.3969/j. issn. 1673-5188. 2017. 02. 005.

[114]. Oikonomou, P., Loukopoulos, T., Dadaliaris, A. N., Koziri, M. G., & Stamoulis, G. I. (2015, October). On formulating and tackling integrated circuit placement as a scheduling problem. In Proceedings of the 19th Panhellenic Conference on Informatics (pp. 86-91). ACM.

[115]. Pandey, S., Wu, L., Guru, S. M., & Buyya, R. (2010, April). A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In Advanced information networking and applications (AINA), 2010 24th IEEE international conference on (pp. 400-407). IEEE.

[116]. Peng, Tian, Wang Huanchen, and Zhang Dongme. "Simulated annealing for the quadratic assignment problem: A further study." Computers & industrial engineering 31.3-4 (1996): 925-928.

[117]. Piñol, P., Migallón, H., López-Granado, O., & Malumbres, M. P. (2015). Slice-based parallel approach for HEVC encoder. The Journal of Supercomputing, 71(5), 1882-1892.

[118]. Puget, Julia Casarin, et al. "Jezz: An effective legalization algorithm for minimum displacement." Proceedings of the 28th Symposium on Integrated Circuits and Systems Design. ACM, 2015.

BIBLIOGRAPHY

[119]. Queyranne, Maurice. "Performance ratio of polynomial heuristics for triangle inequality quadratic assignment problems." Operations Research Letters 4.5 (1986): 231-234.

[120]. Sahni, Sartaj, and Teofilo Gonzalez. "P-complete approximation problems." Journal of the ACM (JACM) 23.3 (1976): 555-565.

[121]. Sait, Sadiq M., Mahmood R. Minhas, and Junaid A. Khan. "Performance and low power driven VLSI standard cell placement using Tabu search." Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on. Vol. 1. IEEE, 2002.

[122]. Salot, P. (2013). A survey of various scheduling algorithm in cloud computing environment. International Journal of Research in Engineering and Technology, 2(2), 131-135.

[123]. Sarkar, V. (1987). Partitioning and scheduling parallel programs for execution on multiprocessors. Stanford Univ., CA (USA).

[124]. Schwarz, H., Marpe, D., & Wiegand, T. (2007). Overview of the scalable video coding extension of the H. 264/AVC standard. IEEE Transactions on circuits and systems for video technology, 17(9), 1103-1120.

[125]. Sechen, Carl, and Alberto Sangiovanni-Vincentelli. "The TimberWolf placement and routing package." IEEE Journal of Solid-State Circuits 20.2 (1985): 510-522.

[126]. Selvarani, S., and G. Sudha Sadhasivam. "Improved cost-based algorithm for task scheduling in cloud computing." Computational intelligence and computing research (iccic), 2010 ieee international conference on. IEEE, 2010.

[127]. Shafique, Muhammad, Muhammad Usman Karim Khan, and Jörg Henkel. "Power efficient and workload balanced tiling for parallelized high efficiency video coding." Image Processing (ICIP), 2014 IEEE International Conference on. IEEE, 2014.

[128]. Spindler, Peter, Ulf Schlichtmann, and Frank M. Johannes. "Abacus: fast legalization of standard cell circuits with minimal movement." Proceedings of the 2008 international symposium on Physical design. ACM, 2008.

[129]. Spindler, Peter, Ulf Schlichtmann, and Frank M. Johannes. "Kraftwerk2—A fast force-directed quadratic placement approach using an accurate net model."

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27.8 (2008): 1398-1411.

[130]. Sridharan, Ramaswami. "The capacitated plant location problem." European Journal of Operational Research 87.2 (1995): 203-213.

[131]. Steinberg, Leon. "The backboard wiring problem: A placement algorithm." Siam Review 3.1 (1961): 37-50.

[132]. Su, S., Li, J., Huang, Q., Huang, X., Shuang, K., & Wang, J. (2013). Cost-efficient task scheduling for executing large programs in the cloud. Parallel Computing, 39(4), 177-188.

[133]. Sullivan, G. J., Ohm, J., Han, W. J., & Wiegand, T. (2012). Overview of the high efficiency video coding (HEVC) standard. IEEE Transactions on circuits and systems for video technology, 22(12), 1649-1668.

[134]. Rapp, Yngve. "Planning of exchange locations and boundaries." Ericsson Technics 2 (1962): 1-22.

[135]. Roy, Jarrod A., et al. "Capo: robust and scalable open-source min-cut floorplacer." Proceedings of the 2005 international symposium on Physical design. ACM, 2005.

[136]. Taghavi, Taraneh, and Xiaojian Yang. "Dragon2005: Large-scale mixed-size placement tool." Proceedings of the 2005 international symposium on Physical design. ACM, 2005.

[137]. Tao, F., Feng, Y., Zhang, L., & Liao, T. W. (2014). CLPS-GA: A case library and Pareto solution-based hybrid genetic algorithm for energy-aware cloud service scheduling. Applied Soft Computing, 19, 264-279.

[138]. Tate, David M., and Alice E. Smith. "A genetic approach to the quadratic assignment problem." Computers & Operations Research 22.1 (1995): 73-83.

[139]. Tawfeek, M. A., El-Sisi, A., Keshk, A. E., & Torkey, F. A. (2013, November). Cloud task scheduling based on ant colony optimization. In Computer Engineering & Systems (ICCES), 2013 8th International Conference on (pp. 64-69). IEEE.

[140]. Teitz, Michael B., and Polly Bart. "Heuristic methods for estimating the generalized vertex median of a weighted graph." Operations research 16.5 (1968): 955-961.

[141]. Terzopoulos, G., & Karatza, H. D. (2016). Power-aware Bag-of-Tasks scheduling on heterogeneous platforms. Cluster Computing, 19(2), 615-631.

[142]. Topcuoglu, H., Hariri, S., & Wu, M. Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE transactions on parallel and distributed systems, 13(3), 260-274.

[143]. Tubular Labs Inc. Available at: https://tubularlabs.com/leaderboards?type=overall&platform=facebook

[144]. Tziritas, N., Khan, S. U., Xu, C. Z., Loukopoulos, T., & Lalis, S. (2013). On minimizing the resource consumption of cloud applications using process migrations. Journal of Parallel and Distributed Computing, 73(12), 1690-1704.

[145]. Vaishnav, Hirendu, and Massoud Pedram. "PCUBE: A performance driven placement algorithm for low power designs." Design Automation Conference, 1993, with EURO-VHDL'93. Proceedings EURO-DAC'93., European. IEEE, 1993.

[146]. VideoLAN. (2017, Jan. 30). x264 home page [Online]. Available: http://www.videolan.org/developers/x264.html

[147]. Viswanathan, Natarajan, Min Pan, and Chris Chu. "FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control." Proceedings of the 2007 Asia and South Pacific Design Automation Conference. IEEE Computer Society, 2007.

[148]. Wah, Benjamin W. "File placement on distributed computer systems." IEEE Computer 17.1 (1984): 23-32.

[149]. Wang, Jia. "A survey of web caching schemes for the internet." ACM SIGCOMM Computer Communication Review 29.5 (1999): 36-46.

[150]. Wang, M., Jayaraman, P. P., Ranjan, R., Mitra, K., Zhang, M., Li, E., ... & Georgeakopoulos, D. (2015). An overview of Cloud based Content Delivery Networks: Research Dimensions and state-of-the-art. In Transactions on Large-Scale Data-and Knowledge-Centered Systems XX (pp. 131-158). Springer Berlin Heidelberg.

BIBLIOGRAPHY

[151]. Wang, X., Wang, Y., & Cui, Y. (2014). A new multi-objective bi-level programming model for energy and locality aware multi-job scheduling in cloud computing. Future Generation Computer Systems, 36, 91-101.

[152]. Wiegand, T., Sullivan, G. J., Bjontegaard, G., & Luthra, A. (2003). Overview of the H. 264/AVC video coding standard. IEEE Transactions on circuits and systems for video technology, 13(7), 560-576.

[153]. Wowza Streaming Cloud. Available at: https://cloud.wowza.com/

[154]. x265 HEVC encoder. Available at: http://x265.org.

[155]. Yang, H., & Tate, M. (2009, December). Where are we at with cloud computing?: a descriptive literature review. In 20th Australasian conference on information systems (pp. 2-4).

[156]. Zakerinasab, M. R., & Wang, M. (2015, October). Dependency-aware distributed video transcoding in the cloud. In Local Computer Networks (LCN), 2015 IEEE 40th Conference on (pp. 245-252). IEEE.

[157]. Zhang, W., Wen, Y., Cai, J., & Wu, D. O. (2014). Toward transcoding as a service in a multimedia cloud: Energy-efficient job-dispatching algorithm. IEEE Transactions on vehicular technology, 63(5), 2002-2012.

[158]. Zhu, N., Rao, L., Liu, X., Liu, J., & Guan, H. (2011, August). Taming power peaks in mapreduce clusters. In ACM SIGCOMM Computer Communication Review (Vol. 41, No. 4, pp. 416-417). ACM.