Systematic Approach to Multi-layer Parallelisation of Time-based Stream Aggregation under Ingest Constraints in the Cloud

THÈSE DE MASTER
présentée le 15 août 2014

à la Faculté informatique et communications
Programme master en informatique
École polytechnique fédérale de Lausanne

et au Laboratoire de media computing
Division recherche et innovation
Technicolor R&D France

pour l'obtention du grade de master ès sciences
par

Bao-Duy TRAN

Membres du jury :

Prof **Rachid GUERRAOUI**, président du jury
Dr **Erwan LE MERRER**, directeur de thèse

Lausanne, Suisse
Rennes, France
2014

On ne voit bien qu'avec le cœur.
L'essentiel est invisible pour les yeux.

— Antoine de Saint Exupéry (1900–1944)

Có công mài sắt, có ngày nên kim.

— Vietnamese proverb

For my Mum

In memoriam my Dad

Acknowledgements

First and foremost, I would like to thank École polytechnique fédérale de Lausanne and Technicolor R&D for providing me with the opportunity to work on this Master's thesis. It has been a challenging yet enjoyable journey. Experience, knowledge and skills obtained through this study will definitely prove beneficial for my future endeavours in the field.

My special gratitude goes to Dr Erwan Le Merrer, my project supervisor, for his dedication and support. Had it not been for his guidance, the achievements in this thesis would not have been possible. I am grateful to Prof Rachid Guerraoui, my academic advisor, for his valuable inputs. This is also an opportunity for me to thank Dr Christoph Neumann, fellow Master's student Grigory Antipov and all other colleagues at the Media Computing Laboratory for their kind support and hospitality.

Last but not least, I am deeply indebted to my dear Mum for her constant love and continuous encouragement throughout the course of this work.

Rennes, 8th August 2014 B.-D. Tran

Abstract

With its real-time capabilities, stream processing is popular for applications like anomaly detection for residential gateways and analytics for business intelligence. Just as other areas of computing, there has been an inevitable trend to shift stream processing to the cloud, thanks to virtualisation technologies and the ubiquity of the Web. Recently launched Amazon Kinesis is amongst cloud-based stream buffer services that bridge the gap between off-cloud sources and cloud-based processing engines. Yet such services are prone to commercial or physical constraints on data ingest rate, calling for the parallelisation and chaining of processing nodes in a multi-layer topology.

In this work, we studied the multi-layer parallelisation of time-based stream aggregation, a commonplace component in stream processing applications, under the impact of ingest rate constraints in the cloud. In particular, comprehensive analyses on rate transfer properties of processing nodes at various aggregation layers were conducted by considering the stream sources (e.g. residential gateways) and their information flow. This led to our proposal of systematic approaches to determining a parallelisation topology that avoids ingest rate saturation while minimising operational costs and deployment complexity. By applying these approaches, system over-provisioning or trial-and-error design can be eliminated.

Our analyses were empirically verified through various simulations. Prototyping in the real Kinesis environment was also conducted to back up our analytical results and proposed topology determination approaches. It is noteworthy that, although the work has been motivated by and prototyped with Amazon Kinesis, it remains generic in nature and its applicability can extend beyond the specific scenario of Kinesis.

Keywords: stream processing; cloud computing; Amazon Kinesis; stream buffer; ingest rate constraint; multi-layer parallelisation; time-based stream aggregation; rate transfer analysis; systematic approach; topology determination; provisioning.

Résumé

Grâce à ses capacités temps-réel, le stream processing (traitement des flux de données) est populaire pour plusieurs applications telles que la détection d'anomalies pour les passerelles résidentielles, ou encore l'analyse pour l'informatique décisionnelle. Comme pour d'autres domaines en informatique, il existe une tendance de fond pour la migration des approches stream processing vers le cloud (nuage informatique), due aux technologies de virtualisation et à l'ubiquité du Web. Amazon Kinesis, récemment lancé, compte parmi les services de tampon pour les flux de données dans le cloud. Ces tampons permettent de relier les sources de données hors-cloud aux moteurs de traitement dans le cloud. Néanmoins, de tels services sont soumis à des contraintes (commerciales ou physiques) sur le débit entrant des données, ce qui rend nécessaire la parallélisation et le chaînage des nœuds de traitement, en une topologie multi-couches.

Dans ce travail, nous avons étudié la parallélisation multi-couches de l'agrégation temporelle des flux de données. C'est une composante dans des applications stream-processing, qui est impactée par les débits d'entrée limités du cloud. En particulier, nous avons effectué une analyse en profondeur des débits résultants sur les nœuds de traitement dans les différentes couches d'agrégation, en considérant les sources (par exemple les passerelles résidentielles) et leurs flux d'information. Cela nous a conduit à proposer des approches systématiques pour la détermination d'une topologie de parallélisation, afin d'éviter la saturation à l'entrée du cloud, tout en minimisant les coûts opérationnels et la complexité de déploiement. En appliquant ces approches, on peut éviter la sur-allocation des ressources et les essais par tâtonnement.

Nos analyses ont été vérifiées empiriquement via des simulations. Des prototypes dans l'environnement Kinesis ont été également effectués pour conforter nos résultats analytiques et nos approches pour la détermination d'une topologie. Il est important de noter que ce travail reste de nature générique et donc que son applicabilité peut s'étendre au-delà du scénario spécifique qui concerne Kinesis d'Amazon, bien que ce dernier en ait constitué sa motivation première, ainsi que son environnement de prototypage.

Mots clefs : stream processing (traitement des flux de données), cloud computing (informatique en nuage), Amazon Kinesis, tampon pour les flux de données, contrainte sur le débit d'entrée, parallélisation multi-couches, agrégation temporelle des flux de données, analyse des débits, approche systématique, détermination d'une topologie, allocation des ressources.

Table of Contents

Acknowledgements **i**

Abstract (English/Français) **iii**

List of Figures **xi**

List of Tables **xiii**

List of Definitions **xv**

List of Theorems **xvii**

List of Remarks **xix**

List of Algorithms **xxi**

List of Symbols **xxiii**

1 Introduction **1**

- 1.1 From batch to stream processing 1
- 1.2 Stream processing systems 2
- 1.3 Streaming towards the cloud 2
- 1.4 Amazon Kinesis 3
- 1.5 Time-based stream aggregation 5
- 1.6 Motivation & problem statement 8
- 1.7 Related work 11
- 1.8 Organisation of the thesis 12

2 Formalisation & methodology **13**

2.1 Project scope 13
2.2 Problem formalisation 14
2.3 Methodology 17

3 Preliminaries for analysis 19

3.1 Sequence 19
3.2 Thinning a sequence 20
3.3 Constant sequence 22
3.4 Binary & alternating binary sequences 23
3.5 Relative rate of Poisson process 27
3.6 Poisson sequence & superposition 27
3.7 Superposition of constant Poisson sequences 31
3.8 Superposition of α-thinned alternating binary Poisson sequences 35

4 Rate transfer analysis 39

4.1 Stream representation for rate analysis 39
4.2 Consideration period 40
4.3 Practical prerequisite at sources 40
4.4 First aggregation layer 42
4.5 Subsequent aggregation layers 44
4.6 Consolidation for all layers 47

5 Topology determination 49

5.1 Problem statement revisited 49
5.2 Road map to topology determination 50
5.3 Priority-based brute force 51
5.4 L-priority mixed-integer non-linear programming 53
5.5 Conservative sequential assignments 55
5.6 L-cap mixed-integer non-linear programming 55

6 Simulations & prototypes 59

6.1 Empirical verification of corollary 3.10 59
6.2 Empirical verification of corollary 3.14 62
6.3 Prototypes of topology determination approaches 64

6.4 Simulation of multi-layer parallelisation 67
6.5 Amazon Kinesis-based prototype 69

7 Concluding remarks 75
7.1 Contributions 75
7.2 Possible future work 76

References 81

Curriculum Vitae (English/Français) 83

List of Figures

1.1 Methods of interaction with Amazon Kinesis 3

1.2 Amazon Kinesis structure and basic operations 4

1.3 Typical usage scenario of Amazon Kinesis 5

1.4 Minimal set-up for time-based stream aggregation 8

1.5 Time-based stream aggregation with single layer of parallelised processors . . . 9

1.6 Time-based stream aggregation with multi-layer parallelisation 10

2.1 Formalised multi-layer parallelisation of time-based stream aggregation 14

3.1 Poisson sequence $\langle u_i \rangle \sim \mathfrak{P}(\lambda)$ as occurrences of u_i at Poisson arrival times τ_i . 28

3.2 Mappings f_s in the superposition of two Poisson sequences 29

3.3 Plot of $y = f(x) = -x^2 + Nx$ on $[1, N-1]$ where $N \in \mathbb{N}^* \setminus \{1\}$ 34

4.1 Consideration periods in time-based stream aggregation with three sources (different colours on timeline denote distinct time period IDs in respective source streams) 40

4.2 Practical scenario with at most two distinct time period IDs in each consideration period (denoted by different colours on timeline of respective source streams) . 41

4.3 Total outgoing rate at first-layer processor as spikes at key-change-moment cluster (colours on timeline denote distinct time period IDs in source streams; diagram not to scale) 44

5.1 Road map to topology determination 51

6.1 Normalised histograms and ideal probability density function of inter-arrival times $\Delta\tau_{ki} \sim \mathsf{Exp}(\lambda_k)$ of controlled Poisson processes $\mathfrak{P}(\lambda_k)$ in the simulation of section 6.1 . . . 60

6.2 Normalised histograms and ideal probability density function of inter-arrival times $\Delta\tau_i \sim \mathsf{Exp}(\lambda)$ and $\Delta\tilde{\tau}_i \sim \mathsf{Exp}(\tilde{\lambda})$ of observed Poisson processes $\mathfrak{P}(\lambda)$ and $\mathfrak{P}(\tilde{\lambda})$ respectively in the simulation of section 6.1 . . . 60

6.3 Arrival rate estimates $\hat{\lambda}_k$, $\hat{\lambda}$ and $\hat{\hat{\lambda}}$ of Poisson processes $\mathfrak{P}(\lambda_k)$, $\mathfrak{P}(\lambda)$ and $\mathfrak{P}(\tilde{\lambda})$ in the simulation of section 6.1, using different look-back window sizes n. For $\hat{\lambda}$ and $\hat{\hat{\lambda}}$, light colour shades indicate 95% confidence intervals. . . . 63

6.4 Normalised histograms and ideal probability density function of inter-arrival times $\Delta\tau_{ki} \sim \mathsf{Exp}(\lambda_k)$ of controlled Poisson processes $\mathfrak{P}(\lambda_k)$ in the simulation of section 6.2 . . . 64

6.5 Normalised histograms and ideal probability density function of inter-arrival times $\Delta\tau_i \sim \mathsf{Exp}(\lambda)$ and $\Delta\tilde{\tau}_i \sim \mathsf{Exp}(\tilde{\lambda})$ of observed Poisson processes $\mathfrak{P}(\lambda)$ and $\mathfrak{P}(\tilde{\lambda})$ respectively in the simulation of section 6.2 . . . 65

6.6 Arrival rate estimates $\hat{\lambda}_k$, $\hat{\lambda}$ and $\hat{\hat{\lambda}}$ of Poisson processes $\mathfrak{P}(\lambda_k)$, $\mathfrak{P}(\lambda)$ and $\mathfrak{P}(\tilde{\lambda})$ in the simulation of section 6.2, using different look-back window sizes n. For $\hat{\lambda}$ and $\hat{\hat{\lambda}}$, light colour shades indicate 95% confidence intervals. . . . 66

6.7 Behaviour at a randomly selected layer-0 source node in the simulation of section 6.4 . . . 68

6.8 Rate estimates at randomly selected layer-ℓ aggregation nodes in the simulation of section 6.4 (look-back window size 200) . . . 69

6.9 Interactions of prototype applications with Amazon Kinesis . . . 70

6.10 Deployment of prototype applications and Amazon Kinesis streams . . . 71

6.11 Percentages of accepted/dropped data in Kinesis-based experiments of section 6.5 – Each experiment is denoted by $\mathbf{n} = [n_\ell]$, numbers of nodes at various aggregation layers . . . 73

List of Tables

3.1 Conditional probabilities given u_1 of alternating binary sequence $\langle u_i \rangle$ 24

6.1 Arrival rate estimates, sample sizes and 95% confidence intervals of rate estimates for homogeneous Poisson processes in the simulation of section 6.1 62

6.2 Arrival rate estimates, sample sizes and 95% confidence intervals of rate estimates for homogeneous Poisson processes in the simulation of section 6.2 65

List of Definitions

1.1 Definition (Time-based stream aggregation) 6

1.2 Definition (Time period ID) 6

1.3 Definition (Target value) 6

1.4 Definition (Admissible aggregate function) 6

1.5 Definition (Time-based stream aggregation output) 7

2.1 Definition (Multi-layer parallelisation) 14

2.2 Definition (Source) 15

2.3 Definition (Aggregation node) 15

2.4 Definition (Processor) 15

2.5 Definition (Sink) 15

2.6 Definition (Rates in multi-layer parallelisation) 16

2.7 Definition (Ingest rate constraint) 16

3.1 Definition (Value set of a sequence) 19

3.2 Definition (Strictly increasing sequence) 19

3.3 Definition (Change index of a sequence) 19

3.4 Definition (Compression of a sequence) 19

3.5 Definition (Sequence thinning) 21

3.6 Definition (Origin function in sequence thinning) 21

3.7 Definition (Constant sequence) 22
3.8 Definition (Characteristic function of constant sequence) 23
3.9 Definition (Binary sequence) 23
3.10 Definition (Alternating binary sequence) 24
3.11 Definition (Unbiased alternating binary sequence) 25
3.12 Definition (Relative mean arrival rate of Poisson process) 27
3.13 Definition (Poisson sequence) 28
3.14 Definition (Superposition of Poisson sequences) 28
3.15 Definition (Source function for Poisson sequence superposition) 29

4.1 Definition (Consideration period) 40

List of Theorems

3.1 Lemma (Constancy between change indices) 20

3.2 Theorem (Consecutive elements of sequence compression) 20

3.3 Lemma (Origin gap in sequence thinning) 22

3.4 Corollary (Compression of binary sequence) 24

3.5 Lemma (Determinism in alternating binary sequence) 24

3.6 Lemma (Prior probability of α-thinned copy of unbiased alternating binary sequence) 25

3.7 Lemma (Consecutive elements in α-thinned copy of alternating binary sequence) 26

3.8 Lemma (Consecutive elements in Poisson sequence superposition) 30

3.9 Theorem (Consecutive elements in superposition of constant Poisson sequences) 31

3.10 Corollary (Events of different consecutive elements in superposition of constant Poisson sequences) 32

3.11 Corollary (Events of different consecutive elements in superposition of constant equi-rate Poisson sequences) 33

3.12 Theorem (Maximum mean arrival rate of Poisson process formed by different-consecutive-elements events in superposition of constant equi-rate Poisson sequences) 34

3.13 Theorem (Consecutive elements in α-thinned alternating binary Poisson sequence superposition) 35

3.14 Corollary (Events of different consecutive elements in α-thinned alternating binary Poisson sequence superposition) 36

3.15 Corollary (Events of different consecutive elements in α-thinned equi-rate alternating binary Poisson sequence superposition) 37

3.16 Theorem (Maximum Poisson mean arrival rate formed by different-consecutive-elements events in α-thinned equi-rate alternating binary Poisson sequence superposition) 37

4.1 Lemma (Input characteristics at first-layer processor) 42

4.2 Theorem (Total incoming rate at first-layer processor) 42

4.3 Theorem (Output characteristics at first-layer processor) 43

4.4 Corollary (Upper bound of total outgoing rate at first-layer processor) 43

4.5 Lemma (Input characteristics at non-first-layer aggregation node) 44

4.6 Theorem (Total incoming rate at non-first-layer aggregation node) 45

4.7 Theorem (Output characteristics at non-first-layer aggregation node) 45

4.8 Corollary (Upper bound of total outgoing rate at non-first-layer aggregation node) 46

4.9 Theorem (Upper bound of total incoming rate at aggregation node) 47

List of Remarks

1.1 Remark (Generalised admissible aggregate function) 7

1.2 Remark (Initialiser & accumulator for admissible aggregate function) 7

3.1 Remark (Value set of sequence compression) 20

3.2 Remark (Sequence thinning using Bernoulli trials) 21

3.3 Remark (1-thinned copy of sequence) 21

3.4 Remark (Value set of α-thinned copy) 21

3.5 Remark (Range of origins in sequence thinning) 21

3.6 Remark (Consecutive elements of constant sequence) 22

3.7 Remark (Thinning of constant sequence) 22

3.8 Remark (Squared characteristic function of constant sequence) 23

3.9 Remark (Sum of characteristic functions of constant sequence) 23

3.10 Remark (Prior probability of constant sequence) 23

3.11 Remark (Alternating binary sequence with value set $\{0,1\}$) 24

3.12 Remark (Thinning of binary sequence) 24

3.13 Remark (Range of Poisson relative mean arrival rate) 27

3.14 Remark (Vector notations for definition 3.12) 27

3.15 Remark (Sum of relative mean arrival rates of Poisson processes) 27

3.16 Remark (Mean arrival rate of Poisson sequence superposition) 28

3.17 Remark (Value set of Poisson sequence superposition) 29

3.18 Remark (Index distances in Poisson sequence superposition) 29

3.19 Remark (Origin in Poisson sequence superposition) 29

3.20 Remark (Independent origins in Poisson sequence superposition) 30

3.21 Remark (Vector notations for theorem 3.9) 31

3.22 Remark (Vector notations for corollary 3.10) 32

3.23 Remark (Upper bound of mean arrival rate of Poisson process formed by different-consecutive-elements events in superposition of constant equi-rate Poisson sequences) 34

3.24 Remark (Vector notations for theorem 3.13) 35

3.25 Remark (Vector notations for corollary 3.14) 36

4.1 Remark (Static system characteristics within a consideration period) 40

List of Algorithms

2.1 Single-buffer, zero-coordination operation at aggregation node 15

3.1 α-thinning of sequence $\langle u_i \rangle$ 21

5.1 Brute-force topology determination: prioritise $\min_{n_1,\dots,n_{L-1}} \sum_{\ell=1}^{L-1} n_\ell$ 52

5.2 Brute-force topology determination: prioritise $\min_L L$, restrict $\sum_{\ell=1}^{L} n_\ell \in \{L..N_{\max}\}$. 53

5.3 MINLP-based topology determination, prioritising $\min_L L$ 54

5.4 Conservative sequential assignments for topology determination 55

5.5 MINLP-based topology determination, restricting $L \in \{1..L_{\max}\}$ ($L_{\max} \in \mathbb{N}^*$) . . . 57

List of Symbols

$\because$	Because, since, as, owing to, due to, etc.
$\therefore$	Therefore, thus, hence, as a result, consequently, etc.
$\varnothing$	Empty set.
$\mathbf{0}$, $\mathbf{1}$	Vector whose elements are all 0's or all 1's.
$\lvert\mathscr{A}\rvert$	Cardinality of set $\mathscr{A}$.
$\mathscr{A}^\star$	Multiset $\mathscr{A}^\star$ (i.e. with element multiplicities).
$\mathscr{A}^\star \uplus \mathscr{B}^\star$	Addition of multisets $\mathscr{A}^\star$ and $\mathscr{B}^\star$.
$(a_1, a_2, \ldots, a_n)$	Ordered tuple of n values ($n \in \mathbb{N}^*$).
$\mathscr{A} \setminus \mathscr{B}$	Relative complement of set $\mathscr{B}$ in set $\mathscr{A}$.
$[a, b]$	Set of reals from a to b, both inclusive ($a, b \in \mathbb{R}$).
$(a, b]$	Set of reals from a exclusive to b inclusive ($a, b \in \mathbb{R}$).
α	Probability parameter for sequence thinning [definition 3.5].
$\mathscr{A}^n$	Cartesian power of set $\mathscr{A}$.
$\mathscr{A} \subseteq \mathscr{B}$ or $\mathscr{B} \supseteq \mathscr{A}$	$\mathscr{A}$ is a subset of $\mathscr{B}$, $\mathscr{B}$ is a superset of $\mathscr{A}$.
$\mathscr{A} \times \mathscr{B}$	Cartesian product of sets $\mathscr{A}$ and $\mathscr{B}$.
$\mathsf{ChiSq}\,(m)$	χ^2 distribution with m degrees of freedom.
C_n^k	Number of all possible unordered combinations of k items chosen out of a set of n items.
Δt	Ideal regular key-change period for sources [definition 2.2].
$\Delta\tau_i$	i^{th} inter-arrival time of a homogeneous Poisson process.
$\Delta\tau_{ki}$	i^{th} inter-arrival time of k^{th} homogeneous Poisson process.
e	Euler's number; $\mathrm{e} = \lim\limits_{n\to\infty}\left(1 + \dfrac{1}{n}\right)^n \approx 2.71828$.
$\mathsf{Exp}\,(\lambda)$	Exponential distribution with rate parameter λ.
$f'(x)$, $f''(x)$	First and second derivatives of function $f(x)$.

IQR $\{x_i\}$	Inter-quartile range of data samples $\{x_i\}$.
L	Number of aggregation layers [definition 2.1].
λ	Mean arrival rate of a homogeneous Poisson process.
λ_0	Mean Poisson emission rate at source [definition 2.2].
$\lambda_{\ell k}^{(\mathrm{in})}(t)$	Total incoming rate at k^{th} node in layer ℓ [definition 2.6].
$\lambda_{(\ell-1)k:\ell m}(t)$	Rate from k^{th} node in layer $\ell - 1$ to m^{th} node in layer ℓ [definition 2.6].
$\lambda_{\ell k}^{(\mathrm{out})}(t)$	Total outgoing rate at k^{th} node in layer ℓ [definition 2.6].
$\hat{m}$	Maximum likelihood estimator of parameter m.
$\mathbb{N}^*$	Set of positive integers, i.e. $\{n \in \mathbb{Z} \mid n > 0\}$.
n_ℓ	Number of layer-ℓ nodes [definition 2.1].
$\{n_{\min}..n_{\max}\}$	Set of integers from $n_{\min}$ to $n_{\max}$ inclusive $(n_{\min}, n_{\max} \in \mathbb{Z})$.
$\mathsf{Normal}\left(\mu, \sigma^2\right)$	Normal distribution with mean μ and variance σ^2.
Ω_k	Random key-change offset at source $(0, k)$ [definition 2.2].
pdf	Abbreviation for '*probability density function*'.
π	Ratio of a circle's circumference to its diameter; $\pi \approx 3.14159$.
$\mathfrak{P}(\lambda)$	Homogeneous Poisson process with mean arrival rate λ.
$p \wedge q$	Logical conjunction of propositions p and q, i.e. p and q.
$p \Leftrightarrow q$	Equivalence of propositions p and q, i.e. p if and only if q.
$\Pr[\mathscr{A}]$	Probability of event $\mathscr{A}$.
$\Pr[\mathscr{A} \mid \mathscr{B}]$	Conditional probability of event $\mathscr{A}$ given event $\mathscr{B}$.
$\psi_\ell(n_1, n_2, \ldots, n_\ell)$	Constraint function used in ℓ^{th} constraint, $\psi_\ell(n_1, n_2, \ldots, n_\ell) \leqslant 0$, for topology determination [section 5.1].
$\mathrm{qf}_{\mathsf{Distrib}}(p)$	Quantile function, i.e. inverse cumulative distribution function, of probability distribution $\mathsf{Distrib}$ $(p \in [0, 1])$.
$\mathbb{R}$	Set of real numbers.
$\mathbb{R}^+$	Set of non-negative reals, i.e. $\{x \in \mathbb{R} \mid x \geqslant 0\}$.
$\mathbb{R}_+^*$	Set of positive reals, i.e. $\{x \in \mathbb{R} \mid x > 0\}$.
ρ_k	Relative mean arrival rate of homogeneous Poisson process $\mathfrak{P}(\lambda_k)$ amongst $\left\{\mathfrak{P}(\lambda_k) \mid k \in \{1..N\}\right\}$ [definition 3.12].
τ_i	Arrival time of the i^{th} event of a homogeneous Poisson process.
T_c	Consideration period [definition 4.1].
θ	Ingest rate constraint at aggregation node [definition 2.7].
$t_i^{(\mathrm{ideal})}$	i^{th} ideal key-change moment for sources [definition 2.2].

t_{ki}	i^{th} key-change moment at source $(0,k)$ [definition 2.2].
$\biguplus_i \mathscr{A}_i^\star$	Addition of multisets $\mathscr{A}_i^\star$.
$\langle u_i \rangle$ or $\langle u_i \rangle_i$	Sequence whose elements u_i are indexed by $i \in \mathbb{N}^*$.
$\bigcup_i \mathscr{A}_i$	Union of sets $\mathscr{A}_i$.
$\langle u_i \rangle \sim \mathfrak{P}(\lambda)$	Poisson sequence [definition 3.13] with mean arrival rate λ.
$\langle u_{ki} \rangle$ or $\langle u_{ki} \rangle_i$	k^{th} sequence whose elements u_{ki} are indexed by $i \in \mathbb{N}^*$.
$\mathbf{v}$	n-dimensional vector $(n \in \mathbb{N}^*)$.
$\langle v_0 \rangle$ or $\langle v_0 \rangle_i$	Constant sequence with value set $\{v_0\}$ [definition 3.7].
$[v]_i$	Vector whose elements are all v.
$[v_i]$ or $[v_i]_i$	Vector whose i^{th} element is v_i.
$\mathbf{v}^\mathsf{T}$	Transpose of vector $\mathbf{v}$.
$\mathbf{v}^\mathsf{T}\mathbf{w}$	Scalar product of vectors $\mathbf{v}$ and $\mathbf{w}$.
$\lVert\mathbf{v}\rVert$	Euclidean norm of vector $\mathbf{v}$.
$\lvert x \rvert$	Absolute value of x.
$\lfloor x \rfloor$, $\lceil x \rceil$	Floor and ceiling of x.
$\xi(v)$	Characteristic function of a constant sequence [definition 3.8].
$\mathbb{Z}$	Set of integers.

1 Introduction

1.1 From batch to stream processing

Since the early days of computer technologies, enterprise and scientific computing has centred around **data processing**. Indeed, the information age has been driven in part by the demand to handle huge datasets. With the advent of modern hardware and the ability to interconnect machines, many breakthroughs have exhibited in the field over the decades.

One major milestone was MapReduce [1] by Google Inc., a simple yet highly scalable model for **batch processing** on clusters of commodity hardware or virtual machines. It requires user-defined `map` and `reduce` phases for key-value data processing while the runtime takes care of data partition, task distribution and scheduling, fault tolerance, and communication. Besides Google's proprietary implementation, Apache Hadoop [2] is a popular open-source MapReduce framework. As the name implies, batch processing engines like Hadoop require data be preloaded into some storage, e.g. a Hadoop Distributed File System (HDFS) [3].

An alternative paradigm called **stream processing** thus emerged to relax the storage requirements and allow 'online' processing upon data arrivals. Not only does it enable processing of gigantic amounts of data at high rate (where store-then-process proves much less feasible), but it is also crucial for applications like infrastructure anomaly detection and real-time business intelligence (where timely insights into live data is vital to competitive advantage).

Various stream processing notions exist in the literature as the model attracted studies at the turn of the century. Babcock et al. [4] defined it as a potentially unbounded data stream of continuous arrivals, no control over processing order and almost no possible retrievals of processed data. Similarly, Stonebraker et al. [5] discussed 'rules' for real-time stream processing — zero-storage processing, in-stream querying, imperfection handling, outcome predictability, high availability, data integration, distribution and scalability, and instantaneous response.

While batch processing has been briefly discussed in this section for background and comparison, we focused solely on stream processing in this work.

1.2 Stream processing systems

Numerous systems have been proposed thus far in **academia**, notably for stream querying like STREAM [6], Aurora [7] and Borealis [8]. In the **industry**, Google Inc. employs MillWheel [9], its low-latency stream processing framework for time-based aggregation, and Photon [10], its distributed system for fault-tolerant and scalable joining of real-time data streams.

In the **open-source** world, Apache Storm [11] is a popular distributed computing framework for scalable, fault-tolerant and real-time processing of data streams. On the whole, Storm deals with the computation of data streamed from sources called *spouts*. It allows to define a topology where these data get processed at computing nodes termed *bolts*. In such a topology, different bolts represent distinct transformations on the data. Each bolt can be scaled as independent and parallel task instances at runtime as the topology is deployed onto a cluster.

1.3 Streaming towards the cloud

Thanks to virtualisation technologies and the ubiquity of the Web, cloud computing is progressing fast with more diverse offerings available, including those from Internet giants like Google Cloud Platform [12] and Amazon Web Services (AWS) [13]. The move from traditional software and infrastructure to online 'utility' services [14, 15] relieves companies of inflexible provisioning and the burden of physical maintenance. Just as other computing areas, stream processing is no exception and the trend to shift towards the cloud is inevitable.

At first glance, the shift could just involve the deployment of any stream processing systems, say Storm [section 1.2], onto cloud-based virtual machines such as Google Compute Engine [16] or Amazon Elastic Compute Cloud (EC2) [17]. Nonetheless, in practice, there remains the issue of ingesting data streams from real-life off-cloud sources. Examples of this 'streaming towards the cloud' scenario include the monitoring of residential gateways by telecommunication operators and the processing of sensor network data streams. This calls for a scalable, reliable and highly available **cloud-based stream buffer** that integrates well with the cloud-enabled stream processing engine of choice, in order to fully realise stream processing in the cloud.

In late 2013, Amazon Web Services Inc. launched **Amazon Kinesis** [18], adding to its cloud computing suite. Much as it is advertised as a service for 'real-time *processing* of streaming data' [18], Kinesis implements precisely and solely the cloud-based stream buffer discussed above with no processing facility provided. In other words, Kinesis must necessarily be employed in conjunction with other systems and services, either from Amazon's cloud ecosystem or otherwise, to complete the cloud-based stream processing work flow.

Section 1.4 attempts to review relevant aspects of Kinesis *beyond* corporate buzzwords and marketing terms, for the service partly serves as the motivation [section 1.6] and prototype environment [section 6.5] for this work. For more comprehensive technical details, the reader is however advised to consult Amazon's official documentation [19].

1.4 Amazon Kinesis

As a Web service, Amazon Kinesis [18] can be interacted via an HTTP Application Programming Interface (**HTTP API**) [20]. Typical clients include Web browsers, cURL [21] and custom programs running an HTTP client library like libcurl [21] or HttpComponents [22]. For convenience, certain administrative tasks can be accomplished through a **Web portal** [23] and Software Development Kits (**SDKs**) [24, 25] are available for Java [26] and PHP [27]. The former embeds relevant calls in Web pages whereas the latter wrap the entire API in the respective languages. Figure 1.1 depicts various methods of interaction with Amazon Kinesis.

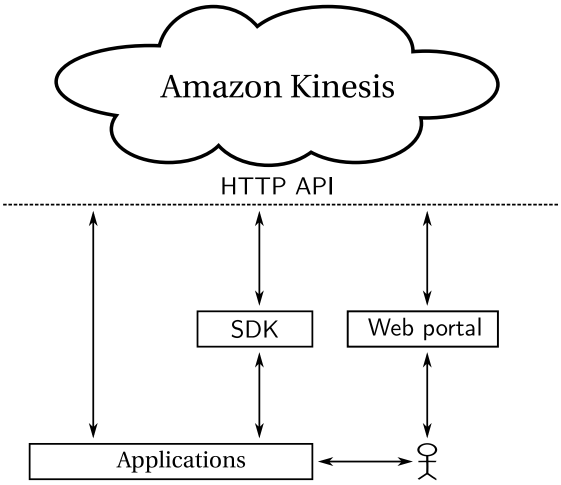


Figure 1.1: Methods of interaction with Amazon Kinesis

In terms of functionality, Kinesis is a high-capacity, high-throughput and reliable buffer in the cloud. It accepts data records from **producers** and releases them to **consumers** upon request. All producer, consumer and dual-role applications are *external* to Kinesis and must be deployed elsewhere, albeit local or remote, physical or cloud-based. Amazon EC2 [17] is thus a viable but not the sole option for hosting Kinesis applications.

Each data **record** comprises a partition key and a free-format payload. It is noteworthy that, due to the nature of HTTP, data ought to be *actively* pushed to and pulled from Kinesis by applications, either via the HTTP API or convenient wrapper SDKs. In other words, no push consumptions are possible by default. Moreover, a particular record can be consumed multiple times by multiple applications; ingested data are actually retained for 24 hours.

The Kinesis buffer of a particular AWS account and region is organised into **streams**. Each stream consists of independent sub-streams called **shards** where data actually reside. Every producer only indicates the desired target stream when pushing data to Kinesis while the aforementioned 'partition key' field is used to route data records to a constituent shard. In fact, a cryptographic hash function is applied on the partition key and each shard is 'in charge' of a portion of the hash value range. Shard splits and merges are possible but can only be specified along the hash value range, which could be less intuitive for certain applications.

Furthermore, each consumer must indicate the specific shard of interest. The service then allows first-in-first-out (FIFO) and last-in-first-out (LIFO) consumption orders on that shard, with FIFO being suitable for many use cases. All in all, the Kinesis buffer structure and its producer-consumer operations can be summarised in the illustration of figure 1.2.

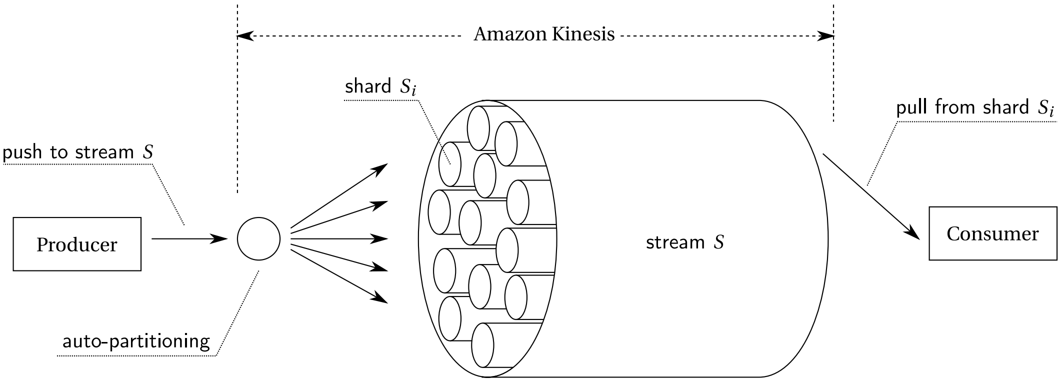


Figure 1.2: Amazon Kinesis structure and basic operations

In Kinesis, **billing** is conducted primarily on the basis of shards and their active duration. At the time of this write-up, each shard costs US\$0.015 an hour. Apart from this data-independent price determined solely by usage set-up and duration, Amazon is also charging another US\$0.028 for every $1,000,000$ records pushed to the service. More details on pricing and payment schemes can be found in Amazon's documentation [28].

One crucial issue with Kinesis is that several **service limits** apply, for technical and/or commercial reasons. Some concern data sizes (e.g. maximum 50 KB for a record payload) and default first-tier offer (e.g. maximum 10 shards in total, extensible on demand). Others pertain to processing throughput, as quoted verbatim from the documentation [29] below:

- Each shard can support up to 5 read transactions per second up to a maximum total of 2 MB of data read per second.
- Each shard can support up to $1,000$ write transactions per second up to a maximum total of 1 MB data written per second.

In the above, 'transaction' refers to an HTTP API call where by design, a write pushes a single record while a read can pull multiple records. These inevitable shard throughput limits and the shard-based billing scheme partly contributed to the motivation of this work [section 1.6].

In practice, most consumer applications would prefer the intuitive push consumption model following the observer design pattern [30, chapter 5] (aka publish-subscribe or listener). Therefore, Amazon offers on top of the SDK a convenient **Kinesis Client Library** (KCL) [31] which implements exactly this *push model*. Meanwhile, KCL is only available for Java [26]. Despite its misleading name, KCL shall not be confused with the SDK. Both are client libraries for Kinesis but the former partially wraps the latter which in turn invokes the underlying HTTP API. Certainly, KCL is an option *only* for consumer applications, and supports *only* the push consumption model. Other operations must still go through the SDK or HTTP API. Indeed, a typical Kinesis usage with producer-consumer applications involves both the SDK and KCL, as shown in figure 1.3, and was considered in this work.

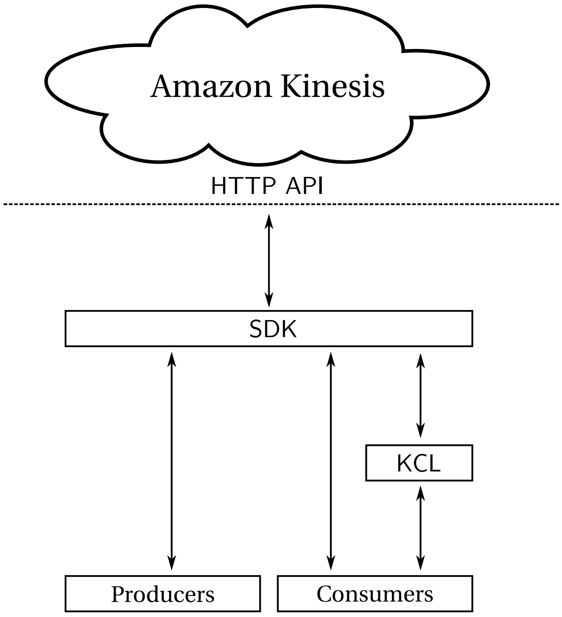


Figure 1.3: Typical usage scenario of Amazon Kinesis

When using KCL, a consumer instance is associated with a stream and can have one (typically) or multiple **workers** instantiated. At most one worker consumes data from each shard at a time. Should the need arises, one worker can be assigned to multiple shards, triggering it to spawn multiple threads, each for a **record processor** responsible for a shard. Record processor is the user-defined listener or event handler that processes data pushed down by KCL from the shard. Assignments of workers to shards are automated by KCL, as opposed to the explicit shard indication when using the SDK. These are coordinated via a checkpointing scheme backed by an Amazon DynamoDB [32] table (hardcoded use by KCL, separate billing). The coordination is scoped to application instances bearing the same application ID.

Last but not least, Kinesis streams and associated application instances can be chained in a **topology** to realise more complex stream processing scenarios. Amazon also offers connector libraries [33, 34] to conveniently integrate Kinesis applications with other Amazon services and Storm [11]; nonetheless, the use of these is not mandatory.

1.5 Time-based stream aggregation

As the title implies, this work examined *time-based stream aggregation*. Despite its simplicity, it is commonplace in many applications, either as a stand-alone one or a constituent phase of a larger data processing work flow. This section introduces its fundamental concepts in order to set the scene for subsequent discussions and analyses.

Definition 1.1 (Time-based stream aggregation). *Time-based stream aggregation* involves multiple streaming data *sources* and an aggregation system which implements an *admissible aggregate function* agg(.) [definition 1.4]. Data sources emit streams of key-value tuples (T, v) where T is a time period ID [definition 1.2] and v is a target value [definition 1.3].

From a particular source, there can be multiple tuples emitted with the same k. Also, the overall collection of tuples with a particular k can originate from many sources. The concrete format of time period IDs is irrelevant as long as they conform to definition 1.2.

Definition 1.2 (Time period ID). In the context of time-based stream aggregation [definition 1.1], *time period ID* T in emitted tuple (T, v) is a unique symbolic identifier drawn from an arbitrary but predefined domain to denote a particular disjoint time period along the universal timeline. Every tuple (T, v) is emitted at a time point contained in the period identified by T.

***Example* 1.1.** In some arbitrary application, $T = 1$ may denote 'August 1985' and $T = 2$ indicates 'September 1985'. In another separate application, $T = \alpha$ can mean '8:00–11:59 am, 8th August 2014' while $T = \beta$ represents '2:00–7:59 pm, 8th August 2014'.

As can be seen from definition 1.2, time period IDs are indeed application-specific but their definitions should be uniform throughout the application concerned. Next, the notion of target value v can be formalised by definition 1.3.

Definition 1.3 (Target value). In the context of time-based stream aggregation [definition 1.1], *target value* v in emitted tuple (T, v) is a data item drawn from an arbitrary but predefined domain, and compatible with admissible aggregate function agg(.) [definition 1.4].

For most practical applications, target value v is usually numerical, for instance, $v \in \mathbb{N}$ or $v \in \mathbb{R}^+$. However, this is not necessarily the case as long as the domain is compatible with the admissible aggregate function in use. Next, we identify the class of admissible aggregate functions considered for time-based stream aggregation by giving definition 1.4.

Definition 1.4 (Admissible aggregate function). Let $\mathcal{V}^\star$ be a multiset of target values [definition 1.3] drawn from domain $\mathscr{D}$. Let $\left\{\mathcal{V}_i^\star \mid i \in \{1..N\}\right\}$ be a partitioning of $\mathcal{V}^\star$ into N disjoint sub-multisets ($N \in \mathbb{N}^*$). Aggregate function $\operatorname{agg}\left(\mathcal{V}^\star\right) \in \mathscr{D}$ is *admissible* if and only if:

$$\operatorname{agg}\left(\mathcal{V}^\star\right) = \operatorname{agg}\left(\biguplus_{i=1}^{N} \left\{\operatorname{agg}\left(\mathcal{V}_i^\star\right)\right\}\right) \tag{1.1}$$

where $\biguplus$ denotes 'addition of multisets'.

***Example* 1.2.** Aggregate functions in relational database management systems (RDBMs) such as `SUM`, `COUNT`, `MAX` and `MIN` are good examples of admissible aggregate functions.

Remark 1.1 (Generalised admissible aggregate function). Definition 1.4 can be generalised with $\text{agg}\left(\mathcal{V}^\star\right)$ returning a multiset of $\mathcal{D}$-valued target values (instead of a single one). Equation (1.1) then becomes:

$$\text{agg}\left(\mathcal{V}^\star\right) = \text{agg}\left(\biguplus_{i=1}^{N} \text{agg}\left(\mathcal{V}_i^\star\right)\right) \tag{1.2}$$

***Example* 1.3.** Examples of generalised agg(.) include finding top-K and bottom-K from a multiset of totally-ordered target values. Note that `MAX` and `MIN` in RDBMs are actually special cases of top-K and bottom-K when $K = 1$.

In this work, only original definition 1.4 was considered, though, without loss of generality.

Remark 1.2 (Initialiser & accumulator for admissible aggregate function). An admissible aggregate function agg(.) can be defined by *initialiser* $\text{agg}(\varnothing)$ (or singleton-based $\text{agg}(\{v\})$) and *accumulator* $\text{agg}\left(\mathcal{V}^\star \uplus \{v\}\right) = \text{accum}_{\text{agg}}\left[\text{agg}\left(\mathcal{V}^\star\right), v\right]$.

***Example* 1.4.** $\text{sum}\left(\mathcal{V}^\star\right) = \sum_{v \in \mathcal{V}^\star} v$ (cf. `SUM` in RDBMs) where $v \in \mathbb{R}, \forall v \in \mathcal{V}^\star$ can be defined by:

$$\begin{cases} \text{sum}(\varnothing) = 0 \text{ or } \text{sum}(\{v\}) = v & \text{(initialiser)} \\ \text{sum}\left(\mathcal{V}^\star \uplus \{v\}\right) = \text{accum}_{\text{sum}}\left[\text{sum}\left(\mathcal{V}^\star\right), v\right] = \text{sum}\left(\mathcal{V}^\star\right) + v & \text{(accumulator)} \end{cases} \tag{1.3}$$

Informally speaking, an aggregate function is admissible if it can be applied in phases – first on disjoint sub-multisets of target values to obtain intermediate results, then on the multiset formed by these intermediate results. In other words, such functions permit partial accumulation and parallelisation of the aggregation process.

Lastly, we conclude this introductory section by presenting the complete time-based stream aggregation work flow in definition 1.5.

Definition 1.5 (Time-based stream aggregation output). In the context of time-based stream aggregation [definition 1.1], for every time period ID T [definition 1.2] present in the streams, the aggregation system must output a single tuple (T, v_T) such that:

$$v_T = \text{agg}\left(\mathcal{V}_T^\star\right) \quad (\forall\, T) \tag{1.4}$$

where $\mathcal{V}_T^\star$ is the multiset of target values v [definition 1.3] emitted in *all* tuples (T, v) from *all* sources, and agg(.) is the admissible aggregate function [definition 1.4] in use.

1.6 Motivation & problem statement

This section explores plausible set-ups for time-based stream aggregation [section 1.5], in general as well as in the cloud context. From there the project motivation is identified through a series of questions then a problem statement is formally presented.

Our goal is to identify a simple yet efficient set-up for time-based stream aggregation. First of all, the straightforward and minimal set-up with a single aggregation node called '**sink**' can be considered [figure 1.4]. Key-value tuples (T, v) from the *sources* are fed to this sink which records T and continuously accumulates v into an internal buffer b using the accumulator of agg(.) [remark 1.2]. Here minimal *single-buffer* processing is adopted. The accumulation continues as long as received tuples have the form $(T, -)$. Upon detecting a key change, i.e. receiving $(T', -)$ where $T' \neq T$ (start of a new time period), the sink outputs (T, b), records T' and reset b using the initialiser of agg(.) [remark 1.2].

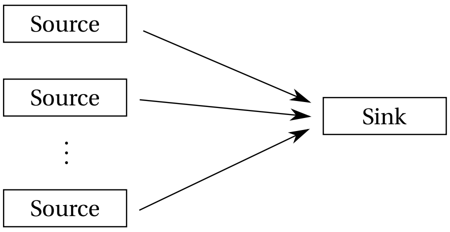


Figure 1.4: Minimal set-up for time-based stream aggregation

In reality, streams from several sources may incur varying propagation delays and slightly different key-change moments due to inaccurate demarcation of time periods. Their interleaving at the sink then results in an 'unclean' output stream where certain keys unsatisfactorily appear multiple times near the ideal point (cf. definition 1.5). To alleviate this, a **cleaner** node can be added after the sink. This cleaner behaves similarly to the sink but it does not output immediately upon a detected key change and delay to a heuristically reasonable extent for 'no' further duplicates. Such delay can be a time-out or based on recent tuples cache, both of which need configuration. In subsequent discussions, a cleaner is assumed to always exist but it will neither be mentioned nor depicted for the sake of brevity.

Figure 1.4's set-up is generally sufficient, unless a substantially large number of sources emit data at considerably fast rates while **ingest rate constraints** exist at aggregation nodes like the sink. Overloaded nodes cannot cope with the incoming amount of data, leading to significant data drops and vastly inaccurate results for time-based stream aggregation. This scenario is highly relevant when porting to the cloud with buffer services like Kinesis imposing throughput limits [section 1.4]. In fact, the sink can be realised by a single-shard Kinesis stream and an associated KCL-based consumer instance with a single worker (consequentially a single record processor). The overall ingest constraint is contributed from the combined read/write limits on the shard, and possibly also bounded computing power at the sink's host.

An intuitive solution to the aforementioned issue is to **parallelise** with multiple aggregation nodes called ‘**processors**’ [figure 1.5]. These processors behave in exactly the same manner as the sink and do not communicate amongst themselves, i.e. *single-buffer, zero-coordination* processing. Outputs from the processors (1^{st} aggregation phase) will finally need to be fed into a single sink (2^{nd} aggregation stage) to ensure a single final output stream. Distribution of data to parallelised nodes should be uniformly random for maximal efficiency and unbiased load balance. Actually, parallelisation and multi-staging are possible thanks to the nature of admissible aggregate function agg(.) [definition 1.4]. Unlike the sink, there is no cleaner attached to the output of every processor for simplicity and efficiency. As cleaners require heuristic configuration of the delay extent and more complex operations, clean-up should be restricted to just before the ultimate output consumption.

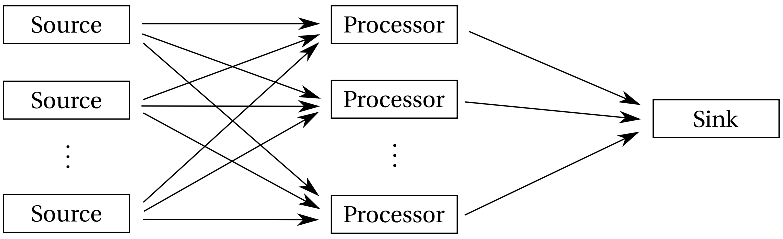


Figure 1.5: Time-based stream aggregation with single layer of parallelised processors

Figure 1.5’s **topology** is also applicable to Kinesis deployment thanks to the ability to chain Kinesis streams and define the number of shards [section 1.4]. In particular, one would have two streams — the first connects the sources to the processors with as many shards as the number of processors while the second links the processors to the sink with one shard. For simplicity and to maximise resource efficiency, one can have as many KCL-based consumer instances (with one internal worker each) as the number of processors so that KCL will assign one full worker for each shard. Effectively, one shard coupled with a full KCL-based single-worker instance implements the model’s abstract processor. Also, for an unbiased uniform spread of data to parallelised processors, Kinesis partition keys [section 1.4] should be randomised; the designation of specific shards for specific data records is undesired here.

A **first question** emerges at this point — ‘*How many processors should there be in the parallelisation layer of figure 1.5?*’ At first glance, the issue might appear straightforward. One may just evaluate the total source emission rate and use that information to compute the number of processors with respect to their ingest constraint (all rates and constraints determined by specifications or empirical assessments). Usually, the more processors, the ‘safer’ it is; however, one needs to balance also the operation costs as more processors imply a higher price to pay. This is specifically true in the case of Kinesis where each processor is associated with one shard, the basis for billing [section 1.4]. An informed safety threshold is therefore preferred, not just going for an infinite number of processors to achieve utmost safety.

In addition, as the whole purpose of introducing parallelisation is to tackle potential problems incurred by ingest rate constraints, a **second question** arises – ‘*Does the single parallelisation layer of figure 1.5 suffice to safely avoid rate-related problems?*’ The answer to this question is no longer as straightforward. In fact, it all depends on the output rate at those processors, which become ingest rate for the subsequent sink. With the presence of real-world propagation delays and inaccurate time period demarcation (see above), the determination of such output rate is in no way trivial and thus requires insightful and streamlined analysis.

Based on the output rates of the first layer, one or more **additional layers** could be necessary before output data can flow into the sink [figure 1.6]. **Two more questions** naturally follow — '*How many additional parallelisation layers are required?*' and '*How many processes should there be in each of these layers?*' Answers to these questions are certainly non-trivial. It is noteworthy that, besides safety with respect to rate saturation issues (which is definitely guaranteed with over-provisioning — infinite number of layers with infinite number of processors each), balancing with the costs directly resulting from the topology size is also of great concern, as discussed above. Again, this is highly relevant when the use of Kinesis is adopted, owing to its inherent service limits and billing basis [section 1.4]. Furthermore, in cases where propagation delays are significant, more layers can introduce unsatisfactory latency as well.

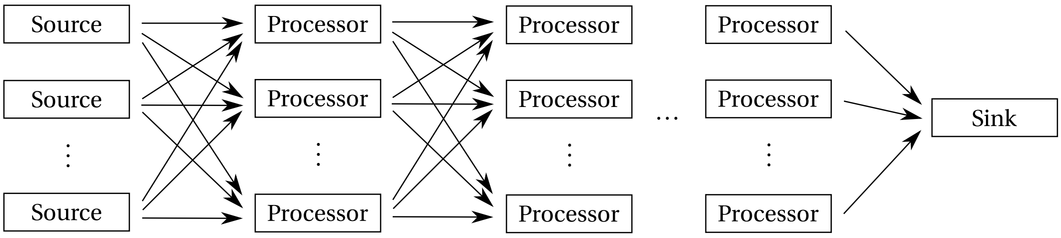


Figure 1.6: Time-based stream aggregation with multi-layer parallelisation

All in all, a systematic approach to determining a safe and reasonable topology for such a multilayer parallelisation is preferable, especially to over-provisioning, purely heuristic estimation or unfounded trial-and-error. Based on the above motivational factors, this work specifically tackled the following problem.

Problem statement. *Considering the multi-layer parallelisation (consisting of aggregation nodes organised as layers of processors and a last aggregation sink, cf. figure 1.6) of time-based stream aggregation [section 1.5] with ingest rate constraints at single-buffer zero-coordination aggregation nodes, derive an informed, streamlined and systematic approach to determine a reasonable parallelisation topology such that ingest rate saturation is guaranteed to be averted at all nodes while minimising the topology size for reduced cost and complexity.*

While partly motivated by cloud-based stream buffer systems like Kinesis [section 1.4], the problem remains sufficiently versatile and generic and could potentially be generalised and extended for applicability in a broader context.

1.7 Related work

As briefly cited in section 1.2, **Photon** [10] is Google's distributed system for fault-tolerant and scalable joining of real-time data streams. It tackles the problem of *joining primary and foreign data streams*, similar to the join operator in relational databases. Operand data however do not reside in tables but are instead presented continuously . Technically speaking, Photon reads timestamped inputs, e.g. event logs, from a distributed Google File System (GFS) [35].

Photon highlights the importance of obtaining joined results in *near real-time* as valuable information with immense *business value* is usually embedded therein. Certain data can be collected in pre-joined form but this could be costly due to traffic overheads. An example is the join between search-query and ad-click streams in Google's advertising system where extracted information allows advertisers to adjust their strategies to evolving user behaviour.

Photon successfully solves the problem under many constraints — exactly-one semantics, fault tolerance at data-centre level, high scalability, low latency, unordered streams and delayed primary streams. Most of these *challenges* are related to having operations across geographically distributed data centres. A notable value is that Photon has been deployed to *real production* environments, permitting thorough performance and design evaluations.

Compared to our work, while sharing the same ultimate goal of achieving timely outputs from data streams, Photon tackles a more specific problem. The focus is rather on how to perform all operations reliably in a distributed and timely manner. Furthermore, Photon also realises a database-like operation (join, instead of aggregation) but it focuses on sophisticated solutions for many distributed computing challenges without considering ingest rate constraints.

In a different realm, **netmap** [36] proposes ultra-fast methods for handling streaming data with close-to-zero latency. Nonetheless, this solution operates at the Media Access Control (MAC) layer or Open Systems Interconnection (OSI) layer 2. Thus it is not directly suitable for application-level problems like time-based stream aggregation presented in this work.

Besides, **ECM-Sketch** [37] approaches stream processing differently. It does sketching of aggregation over distributed, high-dimensional streams using sliding window. It does not deal with ingest rates but employs a tree for aggregation. Our work could therefore be adapted to determine the tree topology for ECM-sketch in the presence of ingest rate constraints.

In terms of application, **LD-Sketch** [38] suggests that time-based stream aggregation is essential for anomaly detection in network traffic data streams. Parallelisation is indeed LD-Sketch's approach to cope with the ever increasing amount and heterogeneity of input streams. Just as ECM-sketch, it still employs the sketching paradigm. Considering Amazon Kinesis [section 1.4] and our motivation discussions [section 1.6], exact aggregation is possible with a processing model such as single-buffer zero-coordination. Hence, the choice between sketching and exact aggregation of streaming data remains an application-specific decision.

1.8 Organisation of the thesis

Apart from this introductory chapter, the rest of the thesis is organised as follows:

- First and foremost, **chapter 2** specifies the project scope with ample discussions and justifications. It then formalises the problem for proper analyses and solutions. The chapter also outlines the methodology adopted in this work.

- Next, **chapter 3** explores several preliminaries necessary for subsequent analyses. Although the contents can always be consulted through back references, the reader is strongly advised to peruse all background information before proceeding.

- **Chapter 4** delves into the details of rate transfer analysis at various layers of the parallelisation of time-based stream aggregation. This analytical chapter and its foundational chapter 3 constitute the most significant portion of the work. Indeed, they together provide a solid basis for the topology determination presented in chapter 5.

- As mentioned above, **chapter 5** is dedicated to topology determination which establishes the systematic approach to multi-layer parallelisation for time-based stream aggregation under ingest constraints — the very topic of this work — based on important results from chapter 4.

- **Chapter 6** presents the empirical verification of certain analyses with simulations, as well as the application of the proposed design approach as a prototype in the real Amazon Kinesis environment.

- Lastly, **chapter 7** concludes the thesis by highlighting major contributions as well as suggesting possible future work.

2 Formalisation & methodology

2.1 Project scope

Owing to the project's time and resource constraints, the problem statement [section 1.6] was tackled within the scope presented here. This does not alter the problem, but rather positions it in acceptable settings for analyses and applications.

First of all, it is assumed that every source emits tuples (T, v) according to a **homogeneous Poisson process** [39, section 2.1, pp. 11–13] on the one-dimensional timeline. Homogeneity refers to the fact that there should be no fluctuations in mean arrival rate. Poisson process is chosen because sources are meant to represent pseudo-regular data arrivals, with minor irregularities due to natural randomness. Indeed, Poisson process has been popular in physics, computer networks and queueing theory to model this kind of arrival phenomena [39].

In addition, all data sources shall be homogeneous amongst themselves, in the sense that they exhibit the **same mean rates**. This assumption works in the application context of time-based stream aggregation whereby multiple sources belong to the same device class configured to emit data in the same manner. Similarly, rates shall be measured in terms of the number of arrivals per second (**items/sec**), implying that all data items have the same size. Bytes-based rates like MB/sec or kbps can be converted to items/sec using an average data item size.

Furthermore, we consider **regular key changes**, implying equal time periods represented by keys k, the time period IDs [definition 1.2]. As a matter of fact, this is a natural set-up for time-based stream aggregation as one typically uses periodic clock- or calendar-based intervals as periods of interest, such as hourly, daily or weekly aggregation. In order to model the inaccuracy of time period demarcations by individual sources, **Gaussian key-change offsets** will be employed. This means actual key change moments are normally distributed around their respective ideally regular ones. This simulates clock skews commonly found in practice, and is analogous to other studies (e.g. Kohno et al. [40] models clock skews using a Gaussian distribution around their server's system time). Since all sources are homogeneous, the aforementioned Gaussian offsets shall have the same variance for consistency.

Last but not least, for the aggregation nodes (processors and the sink), while being completely independent from one another due to zero-coordination processing [section 1.6], all shall exhibit the **same ingest rate constraint**. This models real-world scenario where one would normally launch multiple instances on the same software and hardware specifications, albeit local or cloud-based, for parallelised aggregation nodes. The rate constraints can be derived from published specifications (e.g. service throughput limits indicated by Kinesis documentation [29]) or obtained from empirical assessments on real systems (to cover multiple aspects like computing power and network bottlenecks at the same time).

2.2 Problem formalisation

In this section, we formalise the multi-layer parallelisation [section 1.6] of time-based stream aggregation [definition 1.1]. Various formal concepts and notations presented here serve as the foundation for subsequent system analyses [chapter 4] as well as the derivation of a systematic approach [chapter 5] to determining a topology for such parallelisation.

Definition 2.1 (Multi-layer parallelisation). The *multi-layer parallelisation* of time-based stream aggregation [definition 1.1] is a topology of n_0 *sources* [definition 2.2] ($n_0 \in \mathbb{N}^*$) in layer 0 and L aggregation layers ($L \in \mathbb{N}^*$). Aggregation layer ℓ ($\ell \in \{1..L-1\}$) comprises n_ℓ *processors* [definition 2.4] ($n_\ell \in \mathbb{N}^*$) while the last layer L has a single *sink* [definition 2.5] ($n_L = 1$). The k^{th} node in layer ℓ ($k \in \{1..n_\ell\}, \ell \in \{0..L\}$) is denoted as (ℓ, k). Consecutive layers $\ell - 1$ and ℓ ($\ell \in \{1..L\}$) form a complete bipartite graph where directed edges point from layers $\ell - 1$ to ℓ, representing directed flows of data streams.

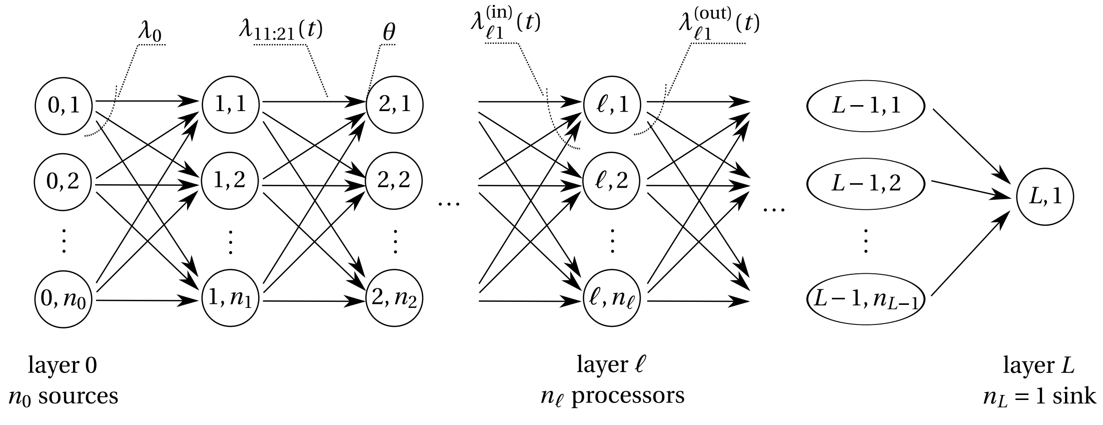


Figure 2.1: Formalised multi-layer parallelisation of time-based stream aggregation

This topology for multi-layer parallelisation is depicted in figure 2.1. Next, we clarify the concepts, functionalities and characteristics of various nodes in the topology.

Definition 2.2 (Source). *Source* $(0, k)$ $(k \in \{1..n_0\})$ in the multi-layer parallelisation [definition 2.1] emits data tuples according to a *homogeneous Poisson process* with same mean arrival rate $\lambda^{(\text{out})}_{0k}(t) = \lambda_0 \in \mathbb{R}^+$ (items/sec), $\forall k \in \{1..n_0\}, \forall t \in \mathbb{R}$. At each source, tuples are uniformly routed to all processors $(1, m)$ in the next layer $(m \in \{1..n_1\})$, i.e. a tuple is routed to processor $(1, m)$ with probability $\frac{1}{n_1}$. Furthermore, at time $t \in \left[t_{ki}, t_{k,i+1}\right]$, source $(0, k)$ emits data tuples in the form (T_{ki}, v). Key-change moment $t_{ki} = t^{(\text{ideal})}_i + \Omega_k$ where independent random offset $\Omega_k \sim \text{Normal}\left(0, \sigma^2_k\right)$ for source $(0, k)$ and $t^{(\text{ideal})}_i = t^{(\text{ideal})}_{i-1} + \Delta t, \forall i$ with constant ideal key-change period $\Delta t \in \mathbb{R}^*_+$ applicable to all sources.

Definition 2.3 (Aggregation node). Every node in layers 1 to L $(L \in \mathbb{N}^*)$ in the multi-layer parallelisation [definition 2.1] is an *aggregation node*, which can either be a processor [definition 2.4] or the sink [definition 2.5].

Definition 2.4 (Processor). *Processor* (ℓ, m) $(\ell \in \{1..L-1\}, m \in \{1..n_\ell\}, L \in \mathbb{N}^*)$ in the multi-layer parallelisation [definition 2.1] receives data tuples in the form of stream superposition from all nodes in the previous layer $\ell - 1$. Each processor runs algorithm 2.1 (single-buffer, zero-coordination operation) using an internal time period ID cache T_0, a target value buffer b (cf. section 1.6), as well as the initialiser and accumulator of admissible aggregation function agg(.) [definition 1.4]. Output tuples are then uniformly routed to all processors $(\ell + 1, w)$ in the next layer $(w \in \{1..n_{\ell+1}\})$, i.e. a tuple is routed to processor $(\ell + 1, w)$ with probability $\frac{1}{n_{\ell+1}}$.

Algorithm 2.1: Single-buffer, zero-coordination operation at aggregation node

```
$T_0 \leftarrow$ null;   $b \leftarrow \text{agg}(\emptyset)$;
foreach tuple $(T, v)$ received do
    if $T_0 = null$ then
        $T_0 \leftarrow T$;   $b \leftarrow \text{agg}(\{v\})$;
    else if $T \neq T_0$ then
        emit tuple $(T_0, b)$;
        $T_0 \leftarrow T$;   $b \leftarrow \text{agg}(\{v\})$;
    else
        $b \leftarrow \text{accum}_{\text{agg}}(b, v)$;
```

Definition 2.5 (Sink). The *sink* $(L, 1)$ $(L \in \mathbb{N}^*)$ in the multi-layer parallelisation [definition 2.1] behaves in precisely the same manner as that of a processor [definition 2.4], except that there is no uniform routing to the next layer (as there is no such layer). Instead, tuples are sent to a *cleaner* node (cf. section 1.6) which interfaces with the ultimate output consumer.

The following definitions formalise the notations of incoming and outgoing rates incurred at various nodes, as well as the ingest rate constraint at these nodes (which is the fundamental condition to be satisfied in our problem statement). These are also illustrated in figure 2.1.

Definition 2.6 (Rates in multi-layer parallelisation). In the context of multi-layer parallelisation [definition 2.1], data stream flowing from nodes $(\ell-1,k)$ to (ℓ,m) $(\ell \in \{1..L\})$ has rate $\lambda_{(\ell-1)k:\ell m}(t)$ (items/sec) at time t. Total incoming rate at time t at aggregation node (ℓ,m) $(\ell \in \{1..L\})$ is denoted as $\lambda^{(\text{in})}_{\ell m}(t)$ (items/sec). Total outgoing rate at time t at source or processor (ℓ,m) $(\ell \in \{0..L-1\})$ is denoted as $\lambda^{(\text{out})}_{\ell m}(t)$ (items/sec).

Definition 2.7 (Ingest rate constraint). Every aggregation node [definition 2.3] in the multi-layer parallelisation [definition 2.1] imposes the same ingest rate constraint θ (items/sec) $(\theta \in \mathbb{R}^+)$. Ingest rate saturation can be avoided (cf. section 1.6) if and only if:

$$\lambda^{(\text{in})}_{\ell m}(t) \leqslant \theta \quad (\forall \ell \in \{1..L\}, \forall m \in \{1..n_\ell\}, \forall t \in \mathbb{R}) \tag{2.1}$$

Lastly, we present the formalised problem statement (cf. section 1.6) using the above notations.

Formalised problem statement. *Consider the formalised multi-layer parallelisation [definition 2.1] of time-based stream aggregation [definition 1.1]. Given the number of sources* n_0*, the total emission rate* λ_0 *at each source, the ingest rate constraint* θ *at each aggregation node, derive an informed, streamlined and systematic approach to determining the parallelisation topology, i.e. the reasonable numbers of processors* n_ℓ *at layers* ℓ $(\ell \in \{1..L-1\})$*, such that ingest rate saturation can be avoided at all aggregation nodes [equation* (2.1)*] while employing the least number of nodes* $\sum_{\ell=1}^{L} n_\ell$ *possible and the least number of layers L possible.*

$$\begin{cases} \min\limits_{n_1,n_2,\ldots,n_{L-1}} \sum\limits_{\ell=1}^{L-1} n_\ell \\ \min\limits_{L} L \\ \text{subject to: } L, n_1, n_2, \ldots, n_{L-1} \in \mathbb{N}^* \\ \text{subject to: } \lambda^{(\text{in})}_{\ell m}(t) \leqslant \theta \quad (\forall \ell \in \{1..L\}, \forall m \in \{1..n_\ell\}, \forall t \in \mathbb{R}) \end{cases} \tag{2.2}$$

It is noteworthy that the determination of the parallelisation topology boils down to determining the numbers of processors n_ℓ at layers ℓ because interconnections within the topology are well-defined (see definition 2.1) and there must always be $n_L = 1$ sink. The minimisation of the number of nodes results in the minimisation of deployment cost (see discussions in section 1.6) while the minimisation of the number of layers reduces complexity and possible latency, consequently increasing overall efficiency.

2.3 Methodology

As can be seen from the formalised problem statement [section 2.2], the ultimate goal is to minimise the total number of aggregation nodes and the total number of layers while satisfying the constraints dictated by equation (2.1). This requires precise knowledge of $\lambda^{(\text{in})}_{\ell m}(t)$, or at least partial knowledge in terms of an upper bound. Indeed, $\lambda^{(\text{in})}_{\ell m}(t)$ at layer ℓ depends on all $\lambda^{(\text{out})}_{(\ell-1)k}(t)$ from the previous layer $\ell - 1$, which in turn depends on all $\lambda_{(\ell-2)w:(\ell-1)k}(t)$ flowing from layer $\ell - 2$ to layer $\ell - 1$, so on and so forth.

As a result, **rate transfer analysis**, which examines the relationship between the above time-varying rates, would be an essential first step. This phase may require several preliminary analyses which are rather general but directly applicable to the rate transfer analysis at hand. Once the rate characteristics at various nodes and layers have been identified, **topology determination** can follow. This phase may require reformulation of the problem statement as well as derivation of concrete solutions. Lastly, empirical verification through **simulations** and **prototyping** in a real environment would help solidify the findings and their applicability.

3 Preliminaries for analysis

As introduced in section 1.8, this chapter may appear to diverge slightly from the problem statement [section 2.2]. However, this is to establish preliminary background necessary for the main discussions in chapters 4 and 5. Definitions and theorems presented here are essential to the principal problem, yet they are generic and can be perused out of context.

Indeed, this chapter forms a solid foundation for subsequent discussions and hence constitutes a major component of the work. In particular, results from sections 3.7 and 3.8 are respectively employed in the rate transfer analyses of the first and subsequent layers of our multi-layer parallelisation [sections 4.4 and 4.5].

3.1 Sequence

First and foremost, we focus on *mathematical sequences* and define the following additional concepts. It will soon become clear as to why sequences are central to our analysis.

Definition 3.1 (Value set of a sequence). $\mathcal{V}$ is a *value set* of sequence $\langle u_i \rangle$ $(i \in \mathbb{N}^*)$ if and only if $\forall\, i \in \mathbb{N}^*, u_i \in \mathcal{V}$ and $\forall\, v \in \mathcal{V}, \exists i \in \mathbb{N}^*$ such that $u_i = v$.

Definition 3.2 (Strictly increasing sequence). Sequence $\langle u_i \rangle$ $(i \in \mathbb{N}^*)$ with a totally ordered value set [definition 3.1] is *strictly increasing* if and only if $\forall\, i_1, i_2 \in \mathbb{N}^*, i_1 < i_2 \Leftrightarrow u_{(i_1)} < u_{(i_2)}$.

Definition 3.3 (Change index of a sequence). Sequence $\langle m_\ell \rangle$ $(\ell \in \mathbb{N}^*)$ with value set $\mathbb{N}^*$ is the *change index* of sequence $\langle u_i \rangle$ $(i \in \mathbb{N}^*)$ if and only if $\langle m_\ell \rangle$ is strictly increasing [definition 3.2] and $\forall\, i \in \mathbb{N}^*, u_i \neq u_{i+1} \Leftrightarrow \exists \ell \in \mathbb{N}^*, m_\ell = i$.

Definition 3.4 (Compression of a sequence). Let $\langle u_i \rangle$ $(i \in \mathbb{N}^*)$ be a sequence with change index $\langle m_\ell \rangle$ $(\ell \in \mathbb{N}^*)$ [definition 3.3]. Sequence $\langle z_\ell \rangle$ $(\ell \in \mathbb{N}^*)$ is the compression of $\langle u_i \rangle$ if and only if $\forall\, \ell \in \mathbb{N}^*, z_\ell = u_{(m_\ell)}$.

Remark 3.1 (Value set of sequence compression). It follows directly from definition 3.4 that the compression of sequence $\langle u_i \rangle$ with value set $\mathcal{V}$ also has value set $\mathcal{V}$.

***Example* 3.1.** Sequence $\langle a, b, b, a, a, a, c, c, b, a, \ldots \rangle$ has value set $\mathcal{V} \supseteq \{a, b, c\}$ [definition 3.1], change index $\langle 1, 3, 6, 8, 9, \ldots \rangle$ [definition 3.3] (which is strictly increasing [definition 3.2]) and compression $\langle a, b, a, c, b, \ldots \rangle$ [definition 3.4] (whose value set is also $\mathcal{V}$ [remark 3.1]).

Next, we present lemma 3.1 pertaining to the constancy of elements between consecutive change index locations. This immediately leads to theorem 3.2 which describes the definite difference between any consecutive elements of a sequence compression [definition 3.4], to be used later in corollary 3.4 [section 3.4].

LEMMA 3.1 (Constancy between change indices). *If $\langle m_\ell \rangle$ is the change index [definition 3.3] of sequence $\langle u_i \rangle$ ($\ell, i \in \mathbb{N}^*$), then $\forall \ell \in \mathbb{N}^*, \forall i \in \left\{(m_\ell + 1) \mathinner{..} \left(m_{(\ell+1)}\right)\right\}, u_i = u_{(m_\ell+1)}$.*

Proof. For $m_{(\ell+1)} = m_\ell + 1$, the proposition holds trivially. We now consider $m_{(\ell+1)} > m_\ell + 1$.

Let $x = u_{(m_\ell+1)}$. For $i = m_\ell + 1$, we have $u_i = x$ by identity. Assuming that the proposition holds for $i = i_0 \in \left\{(m_\ell + 1) \mathinner{..} \left(m_{(\ell+1)} - 1\right)\right\}$, i.e. $u_{(i_0)} = x$, if $u_{(i_0+1)} \neq x$ then $u_{(i_0)} \neq u_{(i_0+1)}$ and $\exists \ell_0 \in \mathbb{N}^*, m_{(\ell_0)} = i_0$ [definition 3.3].

We have $m_{(\ell_0)} = i_0 \in \left\{(m_\ell + 1) \mathinner{..} \left(m_{(\ell+1)} - 1\right)\right\} \Leftrightarrow m_\ell < m_{(\ell_0)} < m_{(\ell+1)}$ but $\langle m_\ell \rangle$ is strictly increasing [definition 3.3] so $\ell < \ell_0 < \ell + 1$ [definition 3.2] $\Leftrightarrow \ell_0 \in \varnothing$, which is contradictory. Hence $u_{(i_0+1)} = x$, i.e. the proposition also holds for $i = i_0 + 1$ given that it holds for $i = i_0$. This concludes our proof by induction for $i \in \left\{(m_\ell + 1) \mathinner{..} \left(m_{(\ell+1)}\right)\right\}$. □

THEOREM 3.2 (Consecutive elements of sequence compression). *If $\langle z_\ell \rangle$ ($\ell \in \mathbb{N}^*$) is the compression [definition 3.4] of some sequence $\langle u_i \rangle$ ($i \in \mathbb{N}^*$), then $\forall \ell \in \mathbb{N}^*, z_\ell \neq z_{\ell+1}$.*

Proof. Let $\langle m_\ell \rangle$ be the change index [definition 3.3] of $\langle u_i \rangle$. If $\exists \ell \in \mathbb{N}^*, z_\ell = z_{\ell+1}$, then $u_{(m_\ell)} = u_{\left(m_{(\ell+1)}\right)}$. However, by definition 3.4, $u_{(m_\ell)} \neq u_{(m_\ell+1)}$ so $u_{(m_\ell+1)} \neq u_{\left(m_{(\ell+1)}\right)}$, which is contradictory with lemma 3.1. Thus the contrary $\forall \ell \in \mathbb{N}^*, z_\ell \neq z_{\ell+1}$ holds. □

3.2 Thinning a sequence

In this section, we formally define the process of creating a new sequence by probabilistically thinning an original one. General properties of the resultant 'thinned' sequence are also discussed. Other properties related to the thinning of specific types of sequences will be explored later in the respective discussions of those sequence types [sections 3.3 and 3.4].

Definition 3.5 (Sequence thinning). Let $\langle u_i \rangle$ ($i \in \mathbb{N}^*$) be a sequence. The *thinning* of $\langle u_i \rangle$ with probability $\alpha \in (0,1]$ (or α-thinning for short) is the process of creating a new sequence $\langle r_j \rangle$ ($j \in \mathbb{N}^*$), termed the *α-thinned copy* of $\langle u_i \rangle$, by transferring every u_i to $\langle r_j \rangle$ with probability α. In other words, u_i is ignored with probability $1-\alpha$. Relative order amongst the elements remains intact, i.e. elements appear in $\langle r_j \rangle$ in the same order as they originally do in $\langle u_i \rangle$.

Remark 3.2 (Sequence thinning using Bernoulli trials). The α-thinning of $\langle u_i \rangle$ [definition 3.5] is equivalent to conducting an independent, identically distributed Bernoulli trial [41, section 3.2] for every $i \in \mathbb{N}^*$, with success probability α (and failure probability $1-\alpha$). If the experiment's outcome is 'success', element u_i gets transferred to $\langle r_j \rangle$; otherwise, it is ignored. The α-thinning of $\langle u_i \rangle$ can thus be expressed as algorithm 3.1 using Bernouilli trials.

Algorithm 3.1: α-thinning of sequence $\langle u_i \rangle$

```
Input: $\langle u_i \rangle$ ($i \in \mathbb{N}^*$), $\alpha \in (0,1]$
Output: $\langle r_j \rangle$ ($j \in \mathbb{N}^*$)
$j \leftarrow 1$;
for $i \leftarrow 1$ to $+\infty$ do
    $X \leftarrow$ Bernoulli-trial($\alpha$);
    if $X = success$ then
        $r_j \leftarrow u_i$; emit $r_j$; $j \leftarrow j+1$;
```

Remark 3.3 (1-thinned copy of sequence). For $\alpha = 1$, there is effectively no thinning as all elements are transferred with absolute certainty. The 1-thinned copy of $\langle u_i \rangle$ is hence a verbatim copy of $\langle u_i \rangle$.

Remark 3.4 (Value set of α-thinned copy). According to definition 3.5, sequence thinning is probabilistic in nature. Any elements from the original sequence $\langle u_i \rangle$ can end up in the α-thinned copy $\langle r_j \rangle$. Thus if $\langle u_i \rangle$ has value set $\mathcal{V}$, then $\langle r_j \rangle$ also has value set $\mathcal{V}$.

To facilitate subsequent analyses, the concept of 'origin' and related properties are now established. Informally speaking, the origin of an element in an α-thinned copy refers to its index in the original sequence.

Definition 3.6 (Origin function in sequence thinning). Let $\langle r_j \rangle$ ($j \in \mathbb{N}^*$) be an α-thinned copy [definition 3.5] of sequence $\langle u_i \rangle$ ($i \in \mathbb{N}^*$). The *origin function* for $\langle r_j \rangle$ is defined as $\text{orig}: \mathbb{N}^* \to \mathbb{N}^*$ where $\text{orig}(j) = i$ if and only if element u_i gets transferred as element r_j during the thinning process.

Remark 3.5 (Range of origins in sequence thinning). As $\mathbb{N}^*$ is the range of probabilistic function orig(.) [definition 3.6] (cf. domain of a random variable), it trivially follows that:

$$\sum_{i=1}^{\infty} \Pr\left[\text{orig}(j) = i\right] = 1 \quad (\forall j \in \mathbb{N}^*) \tag{3.1}$$

Lemma 3.3 (Origin gap in sequence thinning). *Let $\langle r_j \rangle$ $(j \in \mathbb{N}^*)$ be an α-thinned copy [definition 3.5] of sequence $\langle u_i \rangle$ $(i \in \mathbb{N}^*)$. The probability that the origins of two consecutive elements in $\langle r_j \rangle$ differ by a distance of d steps is:*

$$\Pr\left[\mathrm{orig}(j+1) = i+d \,\middle|\, \mathrm{orig}(j) = i\right] = \alpha(1-\alpha)^{d-1} \quad (\forall i, j, d \in \mathbb{N}^*) \tag{3.2}$$

Proof. Consider the α-thinning of sequence $\langle u_i \rangle$ as a series of independent, identically distributed Bernoulli trials [remark 3.2 and algorithm 3.1].

Given that r_j originates from u_i ($\Leftrightarrow \mathrm{orig}(j) = i$, $\forall i, j \in \mathbb{N}^*$), the fact that the next element r_{j+1} originates from u_{i+d} ($\Leftrightarrow \mathrm{orig}(j+1) = i+d$, $\forall d \in \mathbb{N}^*$), which is d steps away from u_i, is equivalent to conducting d additional Bernouilli trials, of which the first $d-1$ fail (hence u_{i+1} to u_{i+d-1} ignored) while the last one succeeds (hence u_{i+d} transferred as r_{j+1}).

As all Bernoulli trials involved have success probability α and failure probability $1-\alpha$ each, and that they are independent from one another, the probability of getting such a series of outcomes ($d-1$ failures followed by a single success) is indeed $\alpha(1-\alpha)^{d-1}$. ☐

3.3 Constant sequence

In this section, we formally define *constant sequence* [definition 3.7], a special type of mathematical sequence in which all elements have the same value. An associated concept called *characteristic function* [definition 3.8] is also established, ready for use in section 3.7.

Definition 3.7 (Constant sequence). Sequence $\langle u_i \rangle$ $(i \in \mathbb{N}^*)$ with value set $\mathcal{V}$ [definition 3.1] is a *constant sequence* if and only if $|\mathcal{V}| = 1$. For $\mathcal{V} = \{v_0\}$, $\langle u_i \rangle$ can be denoted as $\langle v_0 \rangle_i$ or $\langle v_0 \rangle$.

Remark 3.6 (Consecutive elements of constant sequence). It follows directly from definition 3.7 that, if $\langle u_i \rangle$ $(i \in \mathbb{N}^*)$ is a constant sequence, then for an unbiased $\mathbb{N}^*$-valued random index I:

$$\Pr\left[u_{I+1} \neq v \mid u_I = v\right] = 0 \quad (\forall v) \tag{3.3}$$

Remark 3.7 (Thinning of constant sequence). Let $\langle u_i \rangle$ $(i \in \mathbb{N}^*)$ be a constant sequence [definition 3.7] with value set $\{v_0\}$. It follows from remark 3.4 that any α-thinned copy [definition 3.5] of $\langle u_i \rangle$ also has value set $\{v_0\}$. As the singleton value set uniquely defines a constant sequence, all α-thinned copies of $\langle u_i \rangle$ are constant sequences with value set $\{v_0\}$, and actually verbatim copies of $\langle u_i \rangle$.

Definition 3.8 (Characteristic function of constant sequence). *Characteristic function* $\xi(v)$ of constant sequence $\langle v_0 \rangle$ [definition 3.7] is defined as follows:

$$\xi(v) = \begin{cases} 1 & \text{if } v = v_0 \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

Remark 3.8 (Squared characteristic function of constant sequence). It follow directly from definition 3.8 that:

$$\xi^2(v) = \xi(v) \quad (\forall\, v) \tag{3.5}$$

Remark 3.9 (Sum of characteristic functions of constant sequence). It follow directly from definition 3.8 that:

$$\sum_{v \in \mathcal{V}} \xi(v) = \begin{cases} 1 & \text{if } \mathcal{V} \ni v_0 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Remark 3.10 (Prior probability of constant sequence). It follows directly from definitions 3.7 and 3.8 that, if $\langle u_i \rangle$ $(i \in \mathbb{N}^*)$ is a constant sequence, then the prior probability of a randomly selected element taking a particular value v is:

$$\Pr[u_I = v] = \xi(v) \quad (\forall\, v) \tag{3.7}$$

where I is an unbiased $\mathbb{N}^*$-valued random index and $\xi(v)$ is the characteristic function evaluated at v of $\langle u_i \rangle$.

3.4 Binary & alternating binary sequences

This section formalises the notion of *binary sequence* [definition 3.9], another important class of mathematical sequences used in our analyses. Related concepts such as the subclass *alternating binary sequence* [definition 3.10] and relevant properties will also be discussed.

Definition 3.9 (Binary sequence). Sequence $\langle u_i \rangle$ $(i \in \mathbb{N}^*)$ with value set $\mathcal{V}$ [definition 3.1] is a *binary sequence* if and only if $|\mathcal{V}| = 2$.

***Example* 3.2.** $\langle 1, 0, 0, 1, 1, 1, 0, \ldots \rangle$ is the snapshot of the first few elements of a binary sequence with value set $\{0, 1\}$.

Definition 3.10 (Alternating binary sequence). Binary sequence $\langle u_i \rangle$ ($i \in \mathbb{N}^*$) with value set $\mathcal{V}$ [definition 3.9] is an *alternating binary sequence* if and only if $\forall i \in \mathbb{N}^*, u_{i+1} \in \mathcal{V} \setminus \{u_i\}$, or equivalently $\forall i \in \mathbb{N}^*, u_i \neq u_{i+1}$.

Remark 3.11 (Alternating binary sequence with value set $\{0,1\}$). If $\mathcal{V} = \{0,1\}$, then $u_{i+1} \in \mathcal{V} \setminus \{u_i\} \Leftrightarrow u_{i+1} = 1 - u_i, \forall i \in \mathbb{N}^*$.

***Example* 3.3.** $\langle 0,1,0,1,0,1,0,1,\ldots \rangle$ is the snapshot of the first few elements of an alternating binary sequence with value set $\{0,1\}$.

Remark 3.12 (Thinning of binary sequence). It follows directly from definitions 3.9 and 3.10 and remark 3.4 that any α-thinned copy of a binary sequence (alternating or not) is also a binary sequence with the same value set.

COROLLARY 3.4 (Compression of binary sequence). *The compression [definition 3.4] of a binary sequence [definition 3.9] is an alternating binary sequence [definition 3.10].*

Proof. Let $\langle u_i \rangle$ ($i \in \mathbb{N}^*$) be a binary sequence with compression $\langle z_\ell \rangle$ ($\ell \in \mathbb{N}^*$). As $\langle u_i \rangle$ is a binary sequence, $\langle z_\ell \rangle$ is also binary [remark 3.1]. Furthermore, we have $z_\ell \neq z_{\ell+1}, \forall \ell \in \mathbb{N}^*$ [theorem 3.2]. Thus by definition 3.10, $\langle z_\ell \rangle$ is an alternating binary sequence. □

LEMMA 3.5 (Determinism in alternating binary sequence). *If $\langle u_i \rangle$ ($i \in \mathbb{N}^*$) is an alternating binary sequence with value set $\{0,1\}$ [definition 3.10], then the values of all elements u_i are deterministic given that of the initial element u_1:*

$$\begin{cases} u_i = u_1 & \forall i = 2k-1, k \in \mathbb{N}^* \text{ (}i\text{ odd)} \\ u_i = 1 - u_1 & \forall i = 2k, k \in \mathbb{N}^* \text{ (}i\text{ even)} \end{cases} \tag{3.8}$$

In particular, the conditional probabilities of table 3.1 apply.

Table 3.1: Conditional probabilities given u_1 of alternating binary sequence $\langle u_i \rangle$

$i \in \mathbb{N}^*$	$v \in \{0,1\}$	$\tilde{v} \in \{0,1\}$	$\Pr[u_i = v \mid u_1 = \tilde{v}]$
odd	0	0	1
odd	0	1	0
odd	1	0	0
odd	1	1	1
even	0	0	0
even	0	1	1
even	1	0	1
even	1	1	0

Proof. For $k = 1$, equation (3.8) is trivially true:

$$\begin{cases} u_{2k-1} = u_1 = u_1 & \text{(identity)} \\ u_{2k} = u_2 = 1 - u_1 & \text{[remark 3.11]} \end{cases}$$

Assuming equation (3.8) is true for some $k_0 \in \mathbb{N}^*$, i.e. $u_{2k_0-1} = u_1$ and $u_{2k_0} = 1 - u_1$, and using the fact that $u_{i+2} = 1 - u_{i+1} = 1 - (1 - u_i) = u_i, \forall i \in \mathbb{N}^*$ [remark 3.11], then we have the following which concludes our proof by induction on $k \in \mathbb{N}^*$:

$$\begin{cases} u_{2(k_0+1)-1} = u_{(2k_0-1)+2} = u_{2k_0-1} = u_1 \\ u_{2(k_0+1)} = u_{2k_0+2} = u_{2k_0} = 1 - u_1 \end{cases}$$

Table 3.1 is then a verbose expression of equation (3.8) in terms of conditional probabilities which only take values 0 and 1 due to determinism. □

Definition 3.11 (Unbiased alternating binary sequence). Alternating binary sequence $\langle u_i \rangle$ ($i \in \mathbb{N}^*$) [definition 3.10] is an *unbiased alternating binary sequence* if and only if the value of initial element u_1 is random but uniformly distributed on binary value set $\mathcal{V}$:

$$\Pr[u_1 = v] = \frac{1}{2} \quad (\forall v \in \mathcal{V}) \tag{3.9}$$

Lemma 3.6 (Prior probability of α-thinned copy of unbiased alternating binary sequence). *If $\langle u_i \rangle$ ($i \in \mathbb{N}^*$) is an unbiased alternating binary sequence with value set $\{0,1\}$ [definition 3.11] and $\langle r_j \rangle$ ($j \in \mathbb{N}^*$) is an α-thinned copy of $\langle u_i \rangle$ ($\alpha \in (0,1]$) [definition 3.5], then the prior probability of a randomly selected element in $\langle r_j \rangle$ taking a particular value $v \in \{0,1\}$ is:*

$$\Pr\left[r_J = v\right] = \frac{1}{2} \quad (\forall v \in \{0,1\}) \tag{3.10}$$

where J is an unbiased $\mathbb{N}^*$-valued random index.

Proof. Using Bayes' theorem and the orig(.) function [definition 3.6], we have $\forall v \in \{0,1\}$:

$$\begin{aligned} \Pr\left[r_J = v\right] &= \sum_{j=1}^{\infty} \Pr\left[r_J = v \mid J = j\right] \Pr\left[J = j\right] = \sum_{j=1}^{\infty} \Pr\left[r_j = v\right] \Pr\left[J = j\right] \\ &= \sum_{j=1}^{\infty} \Pr\left[J = j\right] \sum_{i=1}^{\infty} \Pr\left[r_j = v \mid \operatorname{orig}(j) = i\right] \Pr\left[\operatorname{orig}(j) = i\right] \\ &= \sum_{j=1}^{\infty} \Pr\left[J = j\right] \sum_{i=1}^{\infty} \Pr[u_i = v] \Pr\left[\operatorname{orig}(j) = i\right] \\ &= \sum_{j=1}^{\infty} \Pr\left[J = j\right] \sum_{i=1}^{\infty} \Pr\left[\operatorname{orig}(j) = i\right] \sum_{\tilde{v}=0}^{1} \Pr[u_i = v \mid u_1 = \tilde{v}] \Pr[u_1 = \tilde{v}] = \frac{1}{2} \end{aligned}$$

$$(\because \text{unbiased}, \Pr[u_1 = \tilde{v}] = \frac{1}{2}, \forall \tilde{v}; \because \text{lemma 3.5}, \sum_{\tilde{v}=0}^{1} \Pr[u_i = v \mid u_1 = \tilde{v}] = 1, \forall i;$$

$$\because \text{remark 3.5}, \sum_{i=1}^{\infty} \Pr\left[\text{orig}(j) = i\right] = 1, \forall j; \because \mathbb{N}^*\text{-valued } J, \sum_{j=1}^{\infty} \Pr\left[J = j\right] = 1) \quad \square$$

Lemma 3.7 (Consecutive elements in α-thinned copy of alternating binary sequence). *If $\langle u_i \rangle$ ($i \in \mathbb{N}^*$) is an alternating binary sequence with value set $\{0,1\}$ [definition 3.11] and $\langle r_j \rangle$ ($j \in \mathbb{N}^*$) is an α-thinned copy of $\langle u_i \rangle$ ($\alpha \in (0,1]$) [definition 3.5], then for an unbiased $\mathbb{N}^*$-valued random index J:*

$$\Pr\left[r_{J+1} = 1 - v \mid r_J = v\right] = \frac{1}{2-\alpha} \quad (\forall v \in \{0,1\}) \tag{3.11}$$

Proof. Due to the determinism of alternating binary sequence $\langle u_i \rangle$ [lemma 3.5]:

$$\forall i, d \in \mathbb{N}^*, \forall v \in \{0,1\}, \quad \Pr[u_{i+d} = 1 - v \mid u_i = v] = \begin{cases} 0 & \text{if } d = 2k, k \in \mathbb{N}^* \ (d \text{ even}) \\ 1 & \text{if } d = 2k-1, k \in \mathbb{N}^* \ (d \text{ odd}) \end{cases} \tag{3.12}$$

Using Bayes' theorem and the orig(.) function [definition 3.6], we have $\forall v \in \{0,1\}$:

$$\Pr\left[r_{J+1} = 1 - v \mid r_J = v\right] = \sum_{j=1}^{\infty} \Pr\left[r_{J+1} = 1 - v \mid r_J = v \wedge J = j\right] \Pr\left[J = j\right]$$

$$= \sum_{i=1}^{\infty} \sum_{d=1}^{\infty} \sum_{j=1}^{\infty} \Pr\left[r_{j+1} = 1 - v \mid r_j = v \wedge \text{orig}(j) = i \wedge \text{orig}(j+1) = i + d\right]$$
$$\Pr\left[J = j\right] \Pr\left[\text{orig}(j) = i \wedge \text{orig}(j+1) = i + d\right]$$

$$= \sum_{i=1}^{\infty} \sum_{d=1}^{\infty} \Pr[u_{i+d} = 1 - v \mid u_i = v] \sum_{j=1}^{\infty} \Pr\left[J = j\right] \Pr\left[\text{orig}(j+1) = i + d \mid \text{orig}(j) = i\right] \Pr\left[\text{orig}(j) = i\right]$$

$$= \sum_{i=1}^{\infty} \sum_{\substack{d=1 \\ d \text{ odd}}}^{\infty} \sum_{j=1}^{\infty} \Pr\left[J = j\right] \alpha (1-\alpha)^{d-1} \Pr\left[\text{orig}(j) = i\right]$$

$$(\because \text{equation (3.12)}; \because \text{lemma 3.3}, \Pr\left[\text{orig}(j+1) = i + d \mid \text{orig}(j) = i\right] = \alpha (1-\alpha)^{d-1}, \forall i, j, d)$$

$$= \alpha \sum_{j=1}^{\infty} \Pr\left[J = j\right] \sum_{i=1}^{\infty} \Pr\left[\text{orig}(j) = i\right] \sum_{k=1}^{\infty} (1-\alpha)^{2(k-1)} \ (\because \text{substituting } d = 2k-1, k \in \mathbb{N}^*)$$

$$= \alpha \sum_{j=1}^{\infty} \Pr\left[J = j\right] \sum_{i=1}^{\infty} \Pr\left[\text{orig}(j) = i\right] \frac{1}{1-(1-\alpha)^2} (\because \text{geometric series with } \left|(1-\alpha)^2\right| < 1)$$

$$= \frac{1}{2-\alpha} \ (\because \alpha \neq 0; \because \text{remark 3.5}, \sum_{i=1}^{\infty} \Pr\left[\text{orig}(j) = i\right] = 1, \forall j; \because \mathbb{N}^*\text{-valued } J, \sum_{j=1}^{\infty} \Pr\left[J = j\right] = 1)$$

□

3.5 Relative rate of Poisson process

It is evident from the problem formalisation [section 2.2] that Poisson processes play a vital role in this work. This section introduces a convenient concept called *relative mean arrival rate*, which is handy when dealing with a set of homogeneous Poisson processes.

Definition 3.12 (Relative mean arrival rate of Poisson process). The *relative mean arrival rate* ρ_k of homogeneous Poisson process $\mathfrak{P}(\lambda_k)$ amongst N homogeneous Poisson processes $\{\mathfrak{P}(\lambda_k) \mid k \in \{1..N\}\}$ $(N \in \mathbb{N}^*)$ with mean arrival rates $\lambda_k \in \mathbb{R}^+, \forall k \in \{1..N\}$ is:

$$\forall k \in \{1..N\}, \quad \rho_k = \begin{cases} \dfrac{1}{N} & \text{if } \lambda_\ell = 0, \forall \ell \in \{1..N\} \\ \dfrac{\lambda_k}{\sum\limits_{\ell=1}^{N} \lambda_\ell} & \text{otherwise} \end{cases} \tag{3.13}$$

Remark 3.13 (Range of Poisson relative mean arrival rate). It follows directly from definition 3.12 that $\rho_k \in [0,1], \forall k \in \{1..N\}, N \in \mathbb{N}^*$.

Remark 3.14 (Vector notations for definition 3.12). If $\boldsymbol{\lambda} = [\lambda_k] \in (\mathbb{R}^+)^N$ and $\mathbf{1} = [1]_k \in \{1\}^N$, then $\forall N \in \mathbb{N}^*$:

$$\boldsymbol{\rho} = [\rho_k] = \begin{cases} \left[\dfrac{1}{N}\right]_k \in \left\{\dfrac{1}{N}\right\}^N & \text{if } \boldsymbol{\lambda} = \mathbf{0} \\ \dfrac{\boldsymbol{\lambda}}{\boldsymbol{\lambda}^\intercal \mathbf{1}} \in [0,1]^N & \text{otherwise} \end{cases} \tag{3.14}$$

Remark 3.15 (Sum of relative mean arrival rates of Poisson processes). It follows directly from definition 3.12 (and the vector notations of remark 3.14) that:

$$\sum_{k=1}^{N} \rho_k = \boldsymbol{\rho}^\intercal \mathbf{1} = 1 \quad (\forall N \in \mathbb{N}^*) \tag{3.15}$$

3.6 Poisson sequence & superposition

This section is dedicated to presenting an extended concept of Poisson process called *Poisson sequence* [definition 3.13]. It also formalises the *superposition* of these sequences [definition 3.14] and discusses related properties.

Definition 3.13 (Poisson sequence). Let $\mathfrak{P}(\lambda)$ be a homogeneous Poisson process with mean arrival rate $\lambda \in \mathbb{R}^+$. Sequence $\langle u_i \rangle$ $(i \in \mathbb{N}^*)$ is a *Poisson sequence* with mean arrival rate λ, denoted as $\langle u_i \rangle \sim \mathfrak{P}(\lambda)$, if and only if there exists a one-to-one correspondence:

$$\begin{aligned} f &: \{\tau_i\} \to \{u_i\} \\ &\quad \tau_i \mapsto u_i, \forall i \in \mathbb{N}^* \end{aligned} \tag{3.16}$$

where $\tau_i \in \mathbb{R}^+$ is the arrival time of the i^{th} event of $\mathfrak{P}(\lambda)$. Such correspondence f describes the Poisson event arriving at time τ_i as the occurrence of element u_i.

***Example* 3.4.** Figure 3.1 illustrates the one-to-one correspondence f which defines a Poisson sequence $\langle u_i \rangle \sim \mathfrak{P}(\lambda)$ as occurrences of elements u_i at Poisson arrival times τ_i [definition 3.13].

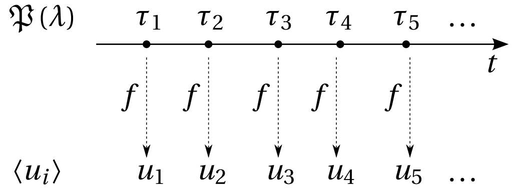


Figure 3.1: Poisson sequence $\langle u_i \rangle \sim \mathfrak{P}(\lambda)$ as occurrences of u_i at Poisson arrival times τ_i

Definition 3.14 (Superposition of Poisson sequences). Let $\{\langle u_{ki} \rangle_i \sim \mathfrak{P}(\lambda_k) \mid k \in \{1..N\}\}$ be N Poisson sequences $(i, N \in \mathbb{N}^*, \lambda_k \in \mathbb{R}^+ \forall k \in \{1..N\})$ [definition 3.13]. Let the superposition of N homogeneous Poisson processes $\{\mathfrak{P}(\lambda_k) \mid k \in \{1..N\}\}$ be $\mathfrak{P}(\lambda)$, which is also a Poisson process [39, Superposition Theorem, section 2.2, pp. 14–17]. Let $\tau_j \in \mathbb{R}^+$ be the arrival time of $\mathfrak{P}(\lambda)$'s j^{th} event $(j \in \mathbb{N}^*)$. There exists a one-to-one correspondence:

$$\begin{aligned} f_s &: \mathbb{N}^* \to \{1..N\} \times \mathbb{N}^* \\ &\quad j \mapsto (k, i) \end{aligned} \tag{3.17}$$

describing every j^{th} event in the superposition $\mathfrak{P}(\lambda)$ as originating from the i^{th} event of constituent process $\mathfrak{P}(\lambda_k)$. Poisson sequence $\langle s_j \rangle \sim \mathfrak{P}(\lambda)$ is the *superposition of Poisson sequences* $\{\langle u_{ki} \rangle \sim \mathfrak{P}(\lambda_k) \mid k \in \{1..N\}\}$ if and only if $s_j = u_{f(j)}, \forall j \in \mathbb{N}^*$.

***Example* 3.5.** Figure 3.2 shows the superposition of two concrete Poisson sequences and illustrates their mappings f_s as given in definition 3.14.

Remark 3.16 (Mean arrival rate of Poisson sequence superposition). According to the Superposition Theorem for Poisson processes [39, section 2.2, pp. 14–17], mean arrival rate λ of Poisson sequence $\langle s_j \rangle \sim \mathfrak{P}(\lambda)$ is:

$$\lambda = \sum_{k=1}^{N} \lambda_k = \boldsymbol{\lambda}^{\mathsf{T}} \mathbf{1} \quad (\forall N \in \mathbb{N}^*) \tag{3.18}$$

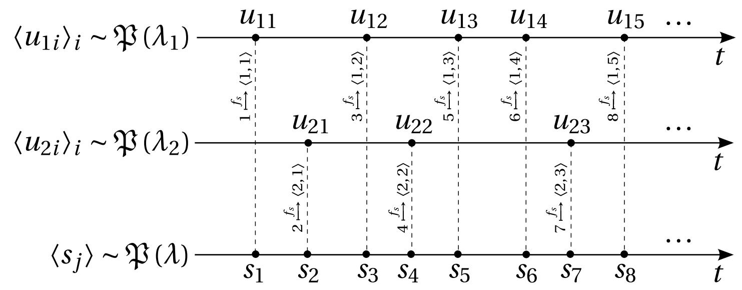

$\langle u_{1i}\rangle_i \sim \mathfrak{P}(\lambda_1)$
u_{11}
u_{12}
u_{13}
u_{14}
u_{15}
…
t
$1 \xrightarrow{f_s} \langle 1,1\rangle$
$3 \xrightarrow{f_s} \langle 1,2\rangle$
$5 \xrightarrow{f_s} \langle 1,3\rangle$
$6 \xrightarrow{f_s} \langle 1,4\rangle$
$8 \xrightarrow{f_s} \langle 1,5\rangle$
$\langle u_{2i}\rangle_i \sim \mathfrak{P}(\lambda_2)$
u_{21}
u_{22}
u_{23}
…
t
$2 \xrightarrow{f_s} \langle 2,1\rangle$
$4 \xrightarrow{f_s} \langle 2,2\rangle$
$7 \xrightarrow{f_s} \langle 2,3\rangle$
$\langle s_j\rangle \sim \mathfrak{P}(\lambda)$
…
s_1
s_2
s_3
s_4
s_5
s_6
s_7
s_8
t

Figure 3.2: Mappings f_s in the superposition of two Poisson sequences

where $\boldsymbol{\lambda} = [\lambda_k] \in (\mathbb{R}^+)^N$ and $\mathbf{1} = [1]_k \in \{1\}^N$.

Remark 3.17 (Value set of Poisson sequence superposition). It is trivial to show that, if constituent $\{\langle u_{ki}\rangle_i \mid k \in \{1..N\}\}$ have value sets $\{\mathcal{V}_k \mid k \in \{1..N\}\}$ [definition 3.1] then the superposition $\langle s_j\rangle$ has value set $\bigcup\limits_{k\in\{1..N\}} \mathcal{V}_k$. Consequently, if all $\{\langle u_{ki}\rangle_i \mid k \in \{1..N\}\}$ are binary sequences each with value set $\mathcal{V}$ [definition 3.9] then $\langle s_j\rangle$ is also a binary sequence with the same value set $\mathcal{V}$.

Remark 3.18 (Index distances in Poisson sequence superposition). Owing to the nature of superposition, elements from each constituent sequence maintain their relative order and retain or increase their relative index distances when appearing in the superposition, forming a partial order in the latter. Employing function f_s given in definition 3.14, the following holds:

$$\forall j_1, j_2, i_1, i_2 \in \mathbb{N}^*, \quad \forall k \in \{1..N\}, \quad \begin{cases} f_s(j_1) = (k, i_1) \\ f_s(j_2) = (k, i_2) \\ j_2 > j_1 \end{cases} \Rightarrow j_2 - j_1 \geqslant i_2 - i_1 > 0 \tag{3.19}$$

Definition 3.15 (Source function for Poisson sequence superposition). Let $\langle s_j\rangle$ be the superposition of N Poisson sequences $\{\langle u_{ki}\rangle_i \mid k \in \{1..N\}\}$ ($N \in \mathbb{N}^*$) [definition 3.14]. The *source function* for $\langle s_j\rangle$ is defined as $\mathrm{src} : \mathbb{N}^* \to \{1..N\}$ where $\mathrm{src}(j) = k$ if and only if element s_j originates from $\langle u_{ki}\rangle_i$, i.e. $\exists i \in \mathbb{N}^*, f_s(j) = (k, i)$ with function f_s given in definition 3.14.

***Example* 3.6.** In the example of figure 3.2, we have $\mathrm{src}(1) = \mathrm{src}(3) = \mathrm{src}(5) = \mathrm{src}(6) = \mathrm{src}(8) = 1$ while $\mathrm{src}(2) = \mathrm{src}(4) = \mathrm{src}(7) = 2$.

Remark 3.19 (Origin in Poisson sequence superposition). With unbiased $\mathbb{N}^*$-valued random index J on $\langle s_j\rangle \sim \mathfrak{P}(\lambda)$, the probability that an element s_J originates from one of independent processes $\{\langle u_{ki}\rangle_i \sim \mathfrak{P}(\lambda_k)\}$ is:

$$\Pr[\mathrm{src}(J) = k] = \rho_k \quad (\forall k \in \{1..N\}, N \in \mathbb{N}^*) \tag{3.20}$$

where ρ_k is the relative mean arrival rate of $\mathfrak{P}(\lambda_k)$ [definition 3.12].

Remark 3.20 (Independent origins in Poisson sequence superposition). In addition to remark 3.19, the origins of distinct elements s_J and s_{J+d} are independent:

$$\Pr[\mathrm{src}(J) = k \wedge \mathrm{src}(J+d) = \ell] = \rho_k \rho_\ell \quad (\forall d \in \mathbb{N}^*, \forall k, \ell \in \{1..N\}, N \in \mathbb{N}^*) \tag{3.21}$$

Next, we establish in lemma 3.8 the probability of two consecutive elements of a binary Poisson sequence superposition being different from each other. This important but general result will be employed in more specific scenarios in theorems 3.9 and 3.13.

LEMMA 3.8 (Consecutive elements in Poisson sequence superposition). *Let $\langle s_j \rangle \sim \mathfrak{P}(\lambda)$ be the superposition of N independent Poisson sequences $\{\langle u_{ki} \rangle_i \sim \mathfrak{P}(\lambda_k) \mid k \in \{1..N\}\}$ [definition 3.14] ($i, j, N \in \mathbb{N}^*$ and $\lambda_k \in \mathbb{R}^+, \forall k \in \{1..N\}$). If $\langle s_j \rangle$ is a binary sequence with value set $\{0,1\}$ [definition 3.9], then the probability of two consecutive elements of $\langle s_j \rangle$ being different from each other is:*

$$\Pr\left[s_J \neq s_{J+1}\right] = \sum_{k=1}^{N} \rho_k \sum_{v=0}^{1} \Pr[u_{kI} = v] \left(\rho_k \Pr\left[u_{k,I+1} = 1 - v \,\middle|\, u_{kI} = v\right] + \sum_{\substack{\ell=1 \\ \ell \neq k}}^{N} \rho_\ell \Pr\left[u_{\ell \tilde{I}} = 1 - v\right] \right) \tag{3.22}$$

where ρ_k is the relative mean arrival rate of homogeneous Poisson process $\mathfrak{P}(\lambda_k)$ [definition 3.12], J is an unbiased $\mathbb{N}^*$-valued random index, and I, $\tilde{I}$ are J-derived random indices such that $(k, I) = f_s(J)$ and $(\ell, \tilde{I}) = f_s(J+1)$ (with function f_s as given in definition 3.14).

Proof. Applying Bayes' theorem and using the facts that $\langle s_j \rangle$ has value set $\{0,1\}$ and $\{\langle u_{ki} \rangle_i\}$ are independent from one another, we have:

$$\begin{aligned}
\Pr\left[s_J \neq s_{J+1}\right] &= \sum_{v=0}^{1} \Pr\left[s_{J+1} = 1 - v \wedge s_J = v\right] \\
&= \sum_{v=0}^{1} \sum_{k=1}^{N} \sum_{\ell=1}^{N} \Pr[\mathrm{src}(J) = k \wedge \mathrm{src}(J+1) = \ell] \Pr\left[s_{J+1} = 1 - v \wedge s_J = v \,\middle|\, \mathrm{src}(J) = k \wedge \mathrm{src}(J+1) = \ell\right] \\
&= \sum_{v=0}^{1} \sum_{k=1}^{N} \sum_{\ell=1}^{N} \rho_k \rho_\ell \Pr\left[u_{\ell \tilde{I}} = 1 - v \wedge u_{kI} = v\right] \quad (\because \text{independent origins [remark 3.20]}) \\
&= \sum_{k=1}^{N} \rho_k \sum_{\ell=1}^{N} \rho_\ell \sum_{v=0}^{1} \Pr[u_{kI} = v] \Pr\left[u_{\ell \tilde{I}} = 1 - v \,\middle|\, u_{kI} = v\right] \\
&= \sum_{k=1}^{N} \rho_k \sum_{v=0}^{1} \Pr[u_{kI} = v] \left(\rho_k \Pr\left[u_{k\tilde{I}} = 1 - v \,\middle|\, u_{kI} = v\right] + \sum_{\substack{\ell=1 \\ \ell \neq k}}^{N} \rho_\ell \Pr\left[u_{\ell \tilde{I}} = 1 - v \,\middle|\, u_{kI} = v\right] \right)
\end{aligned} \tag{3.23}$$

- For $\ell = k$ (i.e. $\mathrm{src}(J) = \mathrm{src}(J+1)$): $f_s(J) = (k, I)$ and $f_s(J+1) = \big(k, \tilde{I}\big)$. According to equation (3.19), $0 < \tilde{I} - I \leqslant (J+1) - J = 1 \Leftrightarrow \tilde{I} = I + 1$.

$$\therefore \Pr\left[u_{k\tilde{I}} = 1 - v \,\middle|\, u_{kI} = v\right] = \Pr\left[u_{k,I+1} = 1 - v \,\middle|\, u_{kI} = v\right] \tag{3.24}$$

- For $l \neq k$: $\langle u_{\ell\tilde{I}} \rangle$ and $\langle u_{kI} \rangle$ are distinct and independent sequences.

$$\therefore \Pr\left[u_{\ell\tilde{I}} = 1 - v \,\middle|\, u_{kI} = v\right] = \Pr\left[u_{\ell\tilde{I}} = 1 - v\right] \tag{3.25}$$

Lastly, substituting equations (3.24) and (3.25) into equation (3.23) yields equation (3.22). □

3.7 Superposition of constant Poisson sequences

The *superposition of constant Poisson sequences* will now be examined. In particular, the **probability** of consecutive elements being different is established [theorem 3.9]. This leads to corollary 3.10 where a closed-form formula is identified for the Poisson **rate** formed by these differences. In corollary 3.11, we narrow down corollary 3.10's scenario to the case where all constituent sequences exhibit the **same rate**. Lastly, theorem 3.12 determines the **upper bounds** of corollary 3.11's rate. These are crucial for the first-layer rate analysis [section 4.4].

THEOREM 3.9 (Consecutive elements in superposition of constant Poisson sequences). *Let Poisson sequence* $\langle s_j \rangle \sim \mathfrak{P}(\lambda)$ *be the superposition of* N *independent Poisson sequences* $\left\{\langle u_{ki} \rangle_i \sim \mathfrak{P}(\lambda_k) \mid k \in \{1..N\}\right\}$ *[definition 3.14] (*$i, j, N \in \mathbb{N}^*$ *and* $\lambda_k \in \mathbb{R}^+, \forall k \in \{1..N\}$*) If every* $\langle u_{ki} \rangle_i$ *is a constant sequence [definition 3.7] with value set* $\mathcal{V}_k$ *such that* $\mathcal{V} = \bigcup\limits_{k\in\{1..N\}} \mathcal{V}_k$ *has cardinality 2 (implying* $N \geqslant 2$*), then the probability of two consecutive elements of* $\langle s_j \rangle$ *being different from each other is:*

$$\Pr\left[s_J \neq s_{J+1}\right] = 1 - \sum_{v\in\mathcal{V}} \left[\sum_{k=1}^{N} \rho_k \xi_k(v)\right]^2 \tag{3.26}$$

where ρ_k *is the relative mean arrival rate of homogeneous Poisson process* $\mathfrak{P}(\lambda_k)$ *[definition 3.12],* $\xi_k(v)$ *is the characteristic function evaluated at* v *of constant sequence* $\langle u_{ki} \rangle_i$ *[definition 3.8], and* J *is an unbiased* $\mathbb{N}^*$*-valued random index.*

Remark 3.21 (Vector notations for theorem 3.9). Using the same vector notations as in remark 3.14 and $\boldsymbol{\xi}_v = [\xi_k(v)]_k \in \{0,1\}^N$, equation (3.26) can be rewritten as:

$$\Pr\left[s_J \neq s_{J+1}\right] = 1 - \sum_{v\in\mathcal{V}} \left(\boldsymbol{\rho}^\intercal \boldsymbol{\xi}_v\right)^2 \tag{3.27}$$

Proof. Superposition $\langle s_j \rangle$ has value set $\bigcup_{k \in \{1..N\}} \mathcal{V}_k = \mathcal{V}$ [remark 3.17]. In addition, $|\mathcal{V}| = 2$ so $\langle s_j \rangle$ is a binary sequence. Without loss of generality, let $\mathcal{V} = \{0,1\}$. Applying lemma 3.8 gives:

$$\Pr\left[s_J \neq s_{J+1}\right] = \sum_{k=1}^{N} \rho_k \sum_{v=0}^{1} \Pr\left[u_{kI} = v\right] \left(\rho_k \Pr\left[u_{k,I+1} = 1 - v \,\middle|\, u_{kI} = v\right] + \sum_{\substack{\ell=1 \\ \ell \neq k}}^{N} \rho_\ell \Pr\left[u_{\ell \tilde{I}} = 1 - v\right] \right)$$

We have the following facts for constant sequences $\{\langle u_{ri} \rangle_i\}$, each with value set $\{0,1\}$:

$$\forall v \in \{0,1\}, \quad \forall r \in \{1..N\}, \quad \begin{cases} \Pr\left[u_{rI} = v\right] = \xi_r(v) & \text{[remark 3.10]} \\ \Pr\left[u_{rI} = 1 - v\right] = 1 - \Pr\left[u_{rI} = v\right] = 1 - \xi_r(v) & \text{[remark 3.10]} \\ \Pr\left[u_{r,I+1} = 1 - v \,\middle|\, u_{rI} = v\right] = 0 & \text{[remark 3.6]} \\ \xi_r(v) = \xi_r^2(v) & \text{[remark 3.8]} \end{cases}$$

$$\text{and also} \begin{cases} \sum_{r=1}^{N} \rho_r = 1 & \text{[remark 3.15]} \\ \sum_{v=0}^{1} \xi_r(v) = 1, \forall r \in \{1..N\} & \because \mathcal{V} = \{0,1\} \supseteq \mathcal{V}_r \ \text{[remark 3.9]} \end{cases}$$

$$\therefore \Pr\left[s_J \neq s_{J+1}\right] = \sum_{k=1}^{N} \rho_k \sum_{v=0}^{1} \xi_k(v) \sum_{\substack{\ell=1 \\ \ell \neq k}}^{N} \rho_\ell \left[1 - \xi_\ell(v)\right]$$

$$= \sum_{k=1}^{N} \rho_k \sum_{v=0}^{1} \xi_k(v) \left\{ -\rho_k \left[1 - \xi_k(v)\right] + \sum_{\ell=1}^{N} \rho_\ell \left[1 - \xi_\ell(v)\right] \right\}$$

$$= -\sum_{k=1}^{N} \rho_k^2 \sum_{v=0}^{1} \xi_k(v) + \sum_{k=1}^{N} \rho_k^2 \sum_{v=0}^{1} \xi_k^2(v) + \sum_{k=1}^{N} \rho_k \sum_{v=0}^{1} \xi_k(v) \sum_{\ell=1}^{N} \rho_\ell - \sum_{k=1}^{N} \rho_k \sum_{v=0}^{1} \xi_k(v) \sum_{\ell=1}^{N} \rho_\ell \xi_\ell(v)$$

$$= -\sum_{k=1}^{N} \rho_k^2 + \sum_{k=1}^{N} \rho_k^2 + 1 - \sum_{v=0}^{1} \sum_{k=1}^{N} \rho_k \xi_k(v) \sum_{\ell=1}^{N} \rho_\ell \xi_\ell(v) = 1 - \sum_{v=0}^{1} \left[\sum_{k=1}^{N} \rho_k \xi_k(v) \right]^2 \qquad \square$$

Corollary 3.10 (Events of different consecutive elements in superposition of constant Poisson sequences). *In the scenario of theorem 3.9, the events of two consecutive elements of* $\langle s_j \rangle$ *being different from each other form a homogeneous Poisson process* $\mathfrak{P}\left(\tilde{\lambda}\right)$ *with mean arrival rate:*

$$\tilde{\lambda} = \left\{ 1 - \sum_{v \in \mathcal{V}} \left[\sum_{k=1}^{N} \rho_k \xi_k(v) \right]^2 \right\} \sum_{\ell=1}^{N} \lambda_\ell \tag{3.28}$$

Remark 3.22 (Vector notations for corollary 3.10). Using the same vector notations as in remark 3.21, equation (3.28) can be rewritten as:

$$\tilde{\lambda} = \left[1 - \sum_{v \in \mathcal{V}} \left(\boldsymbol{\rho}^\intercal \boldsymbol{\xi}_v \right)^2 \right] \boldsymbol{\lambda}^\intercal \mathbf{1} \tag{3.29}$$

Proof. Superposition $\langle s_j \rangle$ has mean arrival rate $\lambda = \sum_{\ell=1}^{N} \lambda_\ell$ [remark 3.16]. Besides, the definition of $\mathfrak{P}\left(\tilde{\lambda}\right)$ implies that it is indeed the thinning of homogeneous Poisson process $\mathfrak{P}(\lambda)$ with probability $\tilde{p} = \Pr[s_{J+1} \neq s_J]$ where J is an unbiased $\mathbb{N}^*$-valued random index. Thus $\mathfrak{P}\left(\tilde{\lambda}\right)$ is also a homogeneous Poisson process with mean arrival rate $\tilde{\lambda} = \tilde{p}\lambda = \tilde{p} \sum_{\ell=1}^{N} \lambda_\ell$ [39, Colouring Theorem, section 5.1, pp. 53–55]. According to theorem 3.9, $\tilde{p} = 1 - \sum_{v \in \mathcal{V}} \left[\sum_{k=1}^{N} \rho_k \xi_k(v) \right]^2$, hence equation (3.28). □

***Example* 3.7.** Let us consider five Poisson sequences $\left\{ \langle u_{ki} \rangle_i \sim \mathfrak{P}(\lambda_k) \mid k \in \{1..5\} \right\}$ satisfying all conditions of corollary 3.10 and having mean arrival rates $\boldsymbol{\lambda} = \begin{bmatrix} 0.1 & 0.3 & 0.35 & 0.15 & 0.1 \end{bmatrix}^\mathsf{T}$ as well as $\boldsymbol{\xi}_0 = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \end{bmatrix}^\mathsf{T}$, $\boldsymbol{\xi}_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \end{bmatrix}^\mathsf{T}$ and $\mathcal{V} = \{0, 1\}$.

Superposition $\langle s_j \rangle$ of $\{\langle u_{ki} \rangle_i\}$ is a Poisson sequence with mean arrival rate $\lambda = \boldsymbol{\lambda}^\mathsf{T}\mathbf{1} = 0.1 + 0.3 + 0.35 + 0.15 + 0.1 = 1$ [remark 3.16]. Relative rates [definition 3.12] of $\left\{\mathfrak{P}(\lambda_k)\right\}$ are $\boldsymbol{\rho} = \dfrac{\boldsymbol{\lambda}}{\boldsymbol{\lambda}^\mathsf{T}\mathbf{1}} = \begin{bmatrix} 0.1 & 0.3 & 0.35 & 0.15 & 0.1 \end{bmatrix}^\mathsf{T}$. Thus $\boldsymbol{\rho}^\mathsf{T}\boldsymbol{\xi}_0 = 0.3 + 0.35 = 0.65$ while $\boldsymbol{\rho}^\mathsf{T}\boldsymbol{\xi}_1 = 0.1 + 0.15 + 0.1 = 0.35$.

Moreover, applying corollary 3.10 in vector notations [remark 3.22] yields mean arrival rate $\tilde{\lambda}$ for the new Poisson process formed by different-consecutive-elements events in $\langle s_j \rangle$: $\tilde{\lambda} = \left[1 - \left(\boldsymbol{\rho}^\mathsf{T}\boldsymbol{\xi}_0\right)^2 - \left(\boldsymbol{\rho}^\mathsf{T}\boldsymbol{\xi}_1\right)^2\right] \boldsymbol{\lambda}^\mathsf{T}\mathbf{1} = \left[1 - (0.65)^2 - (0.35)^2\right] \times 1 = 0.455$.

Corollary 3.11 (Events of different consecutive elements in superposition of constant equi-rate Poisson sequences). *In the scenario of corollary 3.10, if* $\lambda_k = \lambda_0, \forall k \in \{1..N\}, \lambda_0 \in \mathbb{R}^+$, *then:*

$$\tilde{\lambda} = \frac{2N_v(N - N_v)}{N} \lambda_0 \quad (\forall v \in \mathcal{V}) \tag{3.30}$$

where $N_v = |\{k \in \{1..N\} \mid \mathcal{V}_k \ni v\}| \in \{1..(N-1)\}, \forall v \in \mathcal{V}$.

Proof. Using vector notations as in remark 3.22, we have $\boldsymbol{\lambda} = [\lambda_0]_k \in \{\lambda_0\}^N$, so $\lambda = \boldsymbol{\lambda}^\mathsf{T}\mathbf{1} = N\lambda_0$, $\boldsymbol{\rho} = \left[\dfrac{1}{N}\right]_k \in \left\{\dfrac{1}{N}\right\}^N$ and $\forall w \in \mathcal{V}$, $\boldsymbol{\rho}^\mathsf{T}\boldsymbol{\xi}_w = \sum_{\substack{k=1 \\ \mathcal{V}_k \ni w}}^{N} \dfrac{1}{N} = \dfrac{N_w}{N}$. We also have $|\mathcal{V}| = 2$ so $\{N_w \mid w \in \mathcal{V}\} = \{Nv, N - Nv\}, \forall v \in \mathcal{V}$. Applying corollary 3.10 yields $\forall v \in \mathcal{V}$:

$$\begin{aligned} \tilde{\lambda} &= \left[1 - \sum_{w \in \mathcal{V}} \left(\boldsymbol{\rho}^\mathsf{T}\boldsymbol{\xi}_w\right)^2\right] \boldsymbol{\lambda}^\mathsf{T}\mathbf{1} = \left[1 - \sum_{w \in \mathcal{V}} \left(\frac{N_w}{N}\right)^2\right] N\lambda_0 = \left[N - \frac{1}{N} \sum_{w \in \mathcal{V}} N_w^2\right] \lambda_0 \\ &= \left[N - \frac{N_v^2 + (N - N_v)^2}{N}\right] \lambda_0 = \frac{N^2 - N_v^2 - N^2 + 2NN_v - N_v^2}{N} \lambda_0 = \frac{2N_v(N - N_v)}{N} \lambda_0 \end{aligned}$$

□

Theorem 3.12 (Maximum mean arrival rate of Poisson process formed by different-consecutive-elements events in superposition of constant equi-rate Poisson sequences). *In the scenario of corollary 3.11:*

$$\forall v \in \mathcal{V},\quad \begin{cases} \max\limits_{N_v \in \{1..(N-1)\}} \tilde{\lambda} = \begin{cases} \dfrac{N}{2}\lambda_0 & \textit{if } N = 2r, \forall r \in \mathbb{N}^* \textit{ (N even)} \\ \dfrac{(N-1)(N+1)}{2N}\lambda_0 & \textit{if } N = 2r+1, \forall r \in \mathbb{N}^* \textit{ (N odd)} \end{cases} \\ \arg \max\limits_{N_v \in \{1..(N-1)\}} \tilde{\lambda} \in \left\{ \left\lfloor \dfrac{N}{2} \right\rfloor, \left\lceil \dfrac{N}{2} \right\rceil \right\} \end{cases} \tag{3.31}$$

Remark 3.23 (Upper bound of mean arrival rate of Poisson process formed by different-consecutive-elements events in superposition of constant equi-rate Poisson sequences). It is noteworthy from theorem 3.12 that $\forall N \in \mathbb{N}^* \setminus \{1\}$ (both odd and even), $N^2 - 1 < N^2 \Leftrightarrow \dfrac{(N-1)(N+1)}{2N} < \dfrac{N}{2}$, therefore $\tilde{\lambda} \leqslant \dfrac{N}{2}\lambda_0, \forall N_v \in \{1..(N-1)\}$.

Proof. Applying corollary 3.11, we have $\forall v \in \mathcal{V}$:

$$\arg \max_{N_v \in \{1..(N-1)\}} \tilde{\lambda} = \arg \max_{N_v \in \mathcal{D}_g} \frac{2N_v(N-N_v)}{N}\lambda_0 = \arg \max_{N_v \in \mathcal{D}_g} \left(-N_v^2 + NN_v\right) = \arg \max_{N_v \in \mathcal{D}_g} g(N_v)$$

where $\mathcal{D}_g = \{1..(N-1)\}$ and function $g : \mathcal{D}_g \to \mathbb{R}$ is $g(N_v) = -N_v^2 + NN_v$.

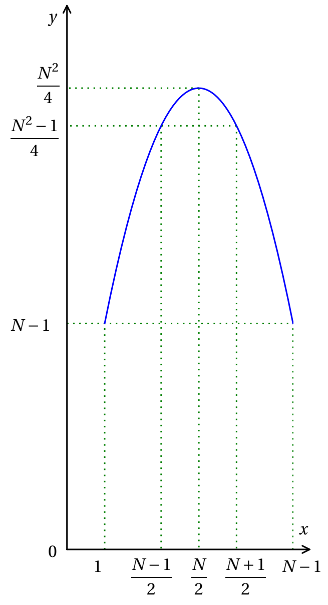


Figure 3.3: Plot of $y = f(x) = -x^2 + Nx$ on $[1, N-1]$ where $N \in \mathbb{N}^* \setminus \{1\}$

Let $\mathcal{D}_f = [1, N-1]$. Analysing function $f : \mathcal{D}_f \to \mathbb{R}$ where $f(x) = -x^2 + Nx$ (plotted in figure 3.3) yields $f'(x) = \dfrac{\mathrm{d}f(x)}{\mathrm{d}x} = -2x + N = 0 \Leftrightarrow x = \dfrac{N}{2}$ and $f''(x) = \dfrac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} = -2 < 0, \forall x \in \mathcal{D}_f$. Thus

$f(x)$ has a single local cum global maximum at $x = \dfrac{N}{2}$. In order to adapt to $g(N_v)$ on $\mathscr{D}_g$, we consider N_v being the closest integers to $\dfrac{N}{2}$, i.e. $\arg\min_{n\in\mathscr{D}_g}\left|n-\dfrac{N}{2}\right| \in \left\{\left\lfloor\dfrac{N}{2}\right\rfloor, \left\lceil\dfrac{N}{2}\right\rceil\right\}$:

- For N even: $\left\lfloor\dfrac{N}{2}\right\rfloor = \left\lceil\dfrac{N}{2}\right\rceil = \dfrac{N}{2}$ so $\arg\max_{N_v\in\mathscr{D}_g}\tilde{\lambda} = \dfrac{N}{2}$, and $\max_{N_v\in\mathscr{D}_g}\tilde{\lambda} = \dfrac{2}{N}\left(\dfrac{N}{2}\right)^2\lambda_0 = \dfrac{N}{2}\lambda_0$.
- For N odd: $\left\lfloor\dfrac{N}{2}\right\rfloor = \dfrac{N-1}{2}$ and $\left\lceil\dfrac{N}{2}\right\rceil = \dfrac{N+1}{2}$ but $g\left(\dfrac{N-1}{2}\right) = g\left(\dfrac{N+1}{2}\right) = \dfrac{N^2-1}{4} = g_0$ so $\arg\max_{N_v\in\mathscr{D}_g}\tilde{\lambda} \in \left\{\left\lfloor\dfrac{N}{2}\right\rfloor, \left\lceil\dfrac{N}{2}\right\rceil\right\}$ and $\max_{N_v\in\mathscr{D}_g}\tilde{\lambda} = \dfrac{2}{N}g_0\lambda_0 = \dfrac{(N-1)(N+1)}{2N}\lambda_0$. □

3.8 Superposition of α-thinned alternating binary Poisson sequences

This section studies the *superposition of α-thinned alternating binary Poisson sequences.* The **probability** of consecutive elements being different is first established in theorem 3.13, then corollary 3.14 derives the Poisson **rate** formed by these differences. Next, corollary 3.15 restricts the scenario of corollary 3.14 to **same-rate** constituent sequences and finally, theorem 3.16 shows the **upper bounds** of corollary 3.14's rate. These results will be crucial for the rate analysis at non-first layers of our multi-layer parallelisation [section 4.5].

THEOREM 3.13 (Consecutive elements in α-thinned alternating binary Poisson sequence superposition). *Let Poisson sequence $\langle s_j\rangle \sim \mathfrak{P}(\lambda)$ be the superposition of N independent Poisson sequences $\left\{\langle u_{ki}\rangle_i \sim \mathfrak{P}(\lambda_k) \mid k \in \{1..N\}\right\}$ [definition 3.14] ($i, j, N \in \mathbb{N}^*$ and $\lambda_k \in \mathbb{R}^+, \forall k \in \{1..N\}$). If every $\langle u_{ki}\rangle_i$ is an α-thinned copy ($\alpha \in (0,1]$) [definition 3.5] of an unbiased alternating binary sequence [definition 3.11] with the same value set $\mathscr{V}$, then the probability of two consecutive elements of $\langle s_j\rangle$ being different from each other is:*

$$\Pr\left[s_J \neq s_{J+1}\right] = \frac{1}{2}\left(1+\frac{\alpha}{2-\alpha}\sum_{k=1}^{N}\rho_k^2\right) \tag{3.32}$$

where ρ_k is the relative mean arrival rate of homogeneous Poisson process $\mathfrak{P}(\lambda_k)$ [definition 3.12] and J is an unbiased $\mathbb{N}^*$-valued random index.

Remark 3.24 (Vector notations for theorem 3.13). Using the same vector notations as in remark 3.14, equation (3.32) can be rewritten as:

$$\Pr\left[s_J \neq s_{J+1}\right] = \frac{1}{2}\left(1+\frac{\alpha}{2-\alpha}\left\|\boldsymbol{\rho}\right\|^2\right) \tag{3.33}$$

Proof. Being an α-thinned copy of a binary sequence with value set $\mathscr{V}$, each $\langle u_{ki}\rangle_i$ is binary with value set $\mathscr{V}$ [remark 3.12], and so is superposition $\langle s_j\rangle$ [remark 3.17]. Without loss of generality, let $\mathcal{V} = \{0, 1\}$. Applying lemma 3.8 gives:

$$\Pr\left[s_J \neq s_{J+1}\right] = \sum_{k=1}^{N} \rho_k \sum_{v=0}^{1} \Pr\left[u_{kI} = v\right] \left(\rho_k \Pr\left[u_{k,I+1} = 1 - v \,\middle|\, u_{kI} = v\right] + \sum_{\substack{\ell=1 \\ \ell \neq k}}^{N} \rho_\ell \Pr\left[u_{\ell \tilde{I}} = 1 - v\right] \right)$$

We have the following facts for α-thinned copies of unbiased alternating binary sequences $\{\langle u_{ki}\rangle_i\}$, each with value set $\{0, 1\}$:

$$\forall v \in \{0,1\}, \quad \forall r \in \{1..N\}, \quad \begin{cases} \Pr\left[u_{rI} = v\right] = \Pr\left[u_{rI} = 1 - v\right] = \dfrac{1}{2}, & \text{[lemma 3.6]} \\ \Pr\left[u_{r,I+1} = 1 - v \,\middle|\, u_{rI} = v\right] = \dfrac{1}{2 - \alpha}, & \text{[lemma 3.7]} \end{cases}$$

$$\text{and also } \sum_{r=1}^{N} \rho_r = 1 \text{ [remark 3.15]}$$

$$\therefore \Pr\left[s_J \neq s_{J+1}\right] = \sum_{k=1}^{N} \rho_k \sum_{v=0}^{1} \frac{1}{2} \left(\frac{\rho_k}{2-\alpha} + \sum_{\substack{\ell=1 \\ \ell \neq k}}^{N} \frac{1}{2} \rho_\ell \right) = \sum_{k=1}^{N} \rho_k \left[\frac{\rho_k}{2-\alpha} + \frac{1}{2} \left(-\rho_k + \sum_{\ell=1}^{N} \rho_\ell \right) \right]$$

$$= \frac{1}{2} \sum_{k=1}^{N} \rho_k \left(\frac{2\rho_k}{2-\alpha} - \rho_k + 1 \right) = \frac{1}{2} \left[\sum_{k=1}^{N} \rho_k + \left(\frac{2}{2-\alpha} - 1 \right) \sum_{k=1}^{N} \rho_k^2 \right] = \frac{1}{2} \left(1 + \frac{\alpha}{2-\alpha} \sum_{k=1}^{N} \rho_k^2 \right) \qquad \square$$

Corollary 3.14 (Events of different consecutive elements in α-thinned alternating binary Poisson sequence superposition). *In the scenario of theorem 3.13, the events of two consecutive elements of $\langle s_j \rangle$ being different from each other form a homogeneous Poisson process $\mathfrak{P}(\tilde{\lambda})$ with mean arrival rate:*

$$\tilde{\lambda} = \frac{1}{2} \left(1 + \frac{\alpha}{2-\alpha} \sum_{k=1}^{N} \rho_k^2 \right) \sum_{\ell=1}^{N} \lambda_\ell \tag{3.34}$$

Remark 3.25 (Vector notations for corollary 3.14). Using the same vector notations as in remark 3.14, equation (3.34) can be rewritten as:

$$\tilde{\lambda} = \frac{1}{2} \left(1 + \frac{\alpha}{2-\alpha} \|\boldsymbol{\rho}\|^2 \right) \boldsymbol{\lambda}^{\mathsf{T}} \mathbf{1} \tag{3.35}$$

Proof. The proof is analogous to that of corollary 3.10 except that theorem 3.13 is applied (instead of theorem 3.9) to obtain $\tilde{p} = \frac{1}{2} \left(1 + \sum_{k=1}^{N} \rho_k^2 \right)$ and arrive at equation (3.34). $\square$

***Example* 3.8.** Let us consider five Poisson sequences $\left\{\langle u_{ki}\rangle_i \sim \mathfrak{P}(\lambda_k) \mid k \in \{1..5\}\right\}$ satisfying all conditions of corollary 3.14 and having mean arrival rates $\boldsymbol{\lambda} = \begin{bmatrix} 0.15 & 0.2 & 0.35 & 0.1 & 0.2 \end{bmatrix}^{\mathsf{T}}$. Also, sequences $\langle u_{ki} \rangle$ are $\frac{1}{3}$-thinned copies of alternating binary sequences, i.e. $\alpha = \frac{1}{3}$.

Superposition $\langle s_j \rangle$ of $\{\langle u_{ki} \rangle_i\}$ is a Poisson sequence with mean arrival rate $\lambda = \boldsymbol{\lambda}^{\intercal}\mathbf{1} = 0.15 + 0.2 + 0.35 + 0.1 + 0.2 = 1$ [remark 3.16]. Relative rates [definition 3.12] of $\left\{\mathfrak{P}(\lambda_k)\right\}$ are $\boldsymbol{\rho} = \dfrac{\boldsymbol{\lambda}}{\boldsymbol{\lambda}^{\intercal}\mathbf{1}} = \begin{bmatrix} 0.15 & 0.2 & 0.35 & 0.1 & 0.2 \end{bmatrix}^{\intercal}$ and $\|\boldsymbol{\rho}\| = \sqrt{(0.15)^2 + (0.2)^2 + (0.35)^2 + (0.1)^2 + (0.2)^2} = \sqrt{0.235}$.

Furthermore, applying corollary 3.14 in vector notations [remark 3.25] yields mean arrival rate $\tilde{\lambda}$ for the new Poisson process formed by different-consecutive-elements events in $\langle s_j \rangle$: $\tilde{\lambda} = \dfrac{1}{2}\left(1 + \dfrac{\alpha}{2-\alpha}\|\boldsymbol{\rho}\|^2\right)\boldsymbol{\lambda}^{\intercal}\mathbf{1} = \dfrac{1}{2}\left(1 + \dfrac{\frac{1}{3}}{2-\frac{1}{3}}\left(\sqrt{0.235}\right)^2\right) \times 1 = 0.5235.$

Corollary 3.15 (Events of different consecutive elements in α-thinned equi-rate alternating binary Poisson sequence superposition). *In the scenario of corollary 3.14, if* $\lambda_k = \lambda_0, \forall k \in \{1..N\}, \lambda_0 \in \mathbb{R}^+$*, then:*

$$\tilde{\lambda} = \frac{1}{2}\left(N + \frac{\alpha}{2-\alpha}\right)\lambda_0 \tag{3.36}$$

Proof. Using vector notations as in remark 3.25, we have $\boldsymbol{\lambda} = [\lambda_0]_k \in \{\lambda_0\}^N$, so $\lambda = \boldsymbol{\lambda}^{\intercal}\mathbf{1} = N\lambda_0$ and $\boldsymbol{\rho} = \left[\dfrac{1}{N}\right]_k \in \left\{\dfrac{1}{N}\right\}^N$, i.e. $\|\boldsymbol{\rho}\|^2 = N \times \left(\dfrac{1}{N}\right)^2 = \dfrac{1}{N}$. Applying corollary 3.14 yields:

$$\tilde{\lambda} = \frac{1}{2}\left(1 + \frac{\alpha}{2-\alpha}\|\boldsymbol{\rho}\|^2\right)\boldsymbol{\lambda}^{\intercal}\mathbf{1} = \frac{1}{2}\left(1 + \frac{\alpha}{2-\alpha} \times \frac{1}{N}\right)N\lambda_0 = \frac{1}{2}\left(N + \frac{\alpha}{2-\alpha}\right)\lambda_0$$

□

Theorem 3.16 (Maximum Poisson mean arrival rate formed by different-consecutive-elements events in α-thinned equi-rate alternating binary Poisson sequence superposition). *In the scenario of corollary 3.15, if* $\lambda_0 \in [0, \lambda_{\max}], \lambda_{\max} \in \mathbb{R}^+$*, then:*

$$\begin{cases} \max\limits_{\lambda_0 \in [0,\lambda_{\max}]} \tilde{\lambda} = \dfrac{1}{2}\left(N + \dfrac{\alpha}{2-\alpha}\right)\lambda_{\max} \\ \arg\max\limits_{\lambda_0 \in [0,\lambda_{\max}]} \tilde{\lambda} = \lambda_{\max} \end{cases} \tag{3.37}$$

Proof. Applying corollary 3.15, we have:

$$\arg\max_{\lambda_0 \in [0,\lambda_{\max}]} \tilde{\lambda} = \arg\max_{\lambda_0 \in [0,\lambda_{\max}]} \frac{1}{2}\left(N + \frac{\alpha}{2-\alpha}\right)\lambda_0 = \arg\max_{\lambda_0 \in [0,\lambda_{\max}]} \lambda_0 = \lambda_{\max}$$

$$\therefore \max_{\lambda_0 \in [0,\lambda_{\max}]} \tilde{\lambda} = \frac{1}{2}\left(N + \frac{\alpha}{2-\alpha}\right)\lambda_{\max}$$

□

4 Rate transfer analysis

At this point, we are ready to shift our focus back to the main problem statement [section 2.2] and delve into the rate transfer analysis at various layers of the parallelisation.

For the purpose of rate analysis, we first derive a simplified data stream representation [section 4.1] and the notion of consideration periods [section 4.2]. We also examine data stream emissions at the sources to state a practical prerequisite [section 4.3]. The actual rate analysis then proceeds layer by layer [sections 4.4 and 4.5], before a consolidation for all layers is presented [section 4.6] ready for use in topology determination [chapter 5].

4.1 Stream representation for rate analysis

Time-based stream aggregation [definition 1.1] deals with streams of data tuples (T, v) comprising time period ID T [definition 1.2] and target value v [definition 1.3]. This is reinforced in the problem formalisation [section 2.2], especially in the definition of data sources [definition 2.2]. Nevertheless, a closer look at aggregation nodes' behaviour [definition 2.3], in particular the operation conducted at these nodes [algorithm 2.1], reveals that a simpler stream representation suffices for rate transfer analysis.

Indeed, rate transfer analysis refers to the determination of an output rate given specific input conditions while the existence of outputs (which governs how the output rate turns out to be) depends solely on time period IDs T [algorithm 2.1]. Therefore, we can consider each data item in our streams (emitted from either a source or an aggregation node) to consist of **only time period ID** T, instead of tuple (T, v). In other words, target values v are irrelevant when it comes to rate analysis, as long as they are aggregated correctly according to algorithm 2.1 using aggregate function agg(.) [definition 1.4].

We shall consistently adopt this simplified stream representation throughout this chapter, hence the interchangeable use of '*data item*' and '*time period ID*' and the characterisation of data streams solely based on the time period IDs contained therein.

4.2 Consideration period

With the simplified stream representation of section 4.1 in place, the characteristics of our system (for the purpose of rate analysis) are determined solely by which time period IDs the sources emit and how they emit these (in terms of rate and other properties). Therefore, we now establish the following important notion of '*consideration period*'.

Definition 4.1 (Consideration period). If we consider the superposition of all key-change moments t_{ki} of all sources $(0,k)$ [definition 2.2] in time-based stream aggregation [definition 1.1], the universal timeline can be split into several *consideration periods* demarcated by these key-change moments.

The illustration in figure 4.1 shows consideration periods for time-based stream aggregation with three sources $(0,1)$, $(0,2)$ and $(0,3)$ during a short time snippet.

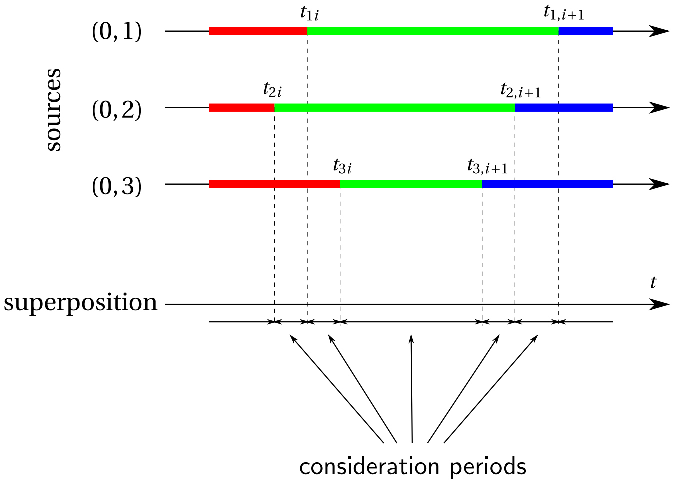


Figure 4.1: Consideration periods in time-based stream aggregation with three sources (different colours on timeline denote distinct time period IDs in respective source streams)

Remark 4.1 (Static system characteristics within a consideration period). It is evident that within each consideration period [definition 4.1], system characteristics remain static with every source $(0,k)$ emitting a *constant Poisson sequence* $\langle u_{ki} \rangle_i \sim \mathfrak{P}(\lambda_0)$ [definitions 3.7 and 3.13] (cf. definition 2.2 and figure 4.1).

4.3 Practical prerequisite at sources

According to definition 2.2, key-change moments at source $(0,k)$ always exhibit random offset Ω_k from the ideal time point determined by constant ideal key-change period Δt. Depending on variance σ_k^2 of normally distributed Ω_k, it can happen, though rather unlikely, that an arbitrary number of distinct time period IDs can be present in a particular consideration period [definition 4.1].

In practice, however, as offsets are typically small (with respect to Δt) causing actual key-change moments to cluster around their respective ideal counterpart, rather than spreading infinitely on both sides of the latter. In such a **practical scenario**, there are *at most two distinct time period IDs* emitted by any sources within a consideration period. We shall adopt this practical scenario as a *prerequisite* for our analyses in this work.

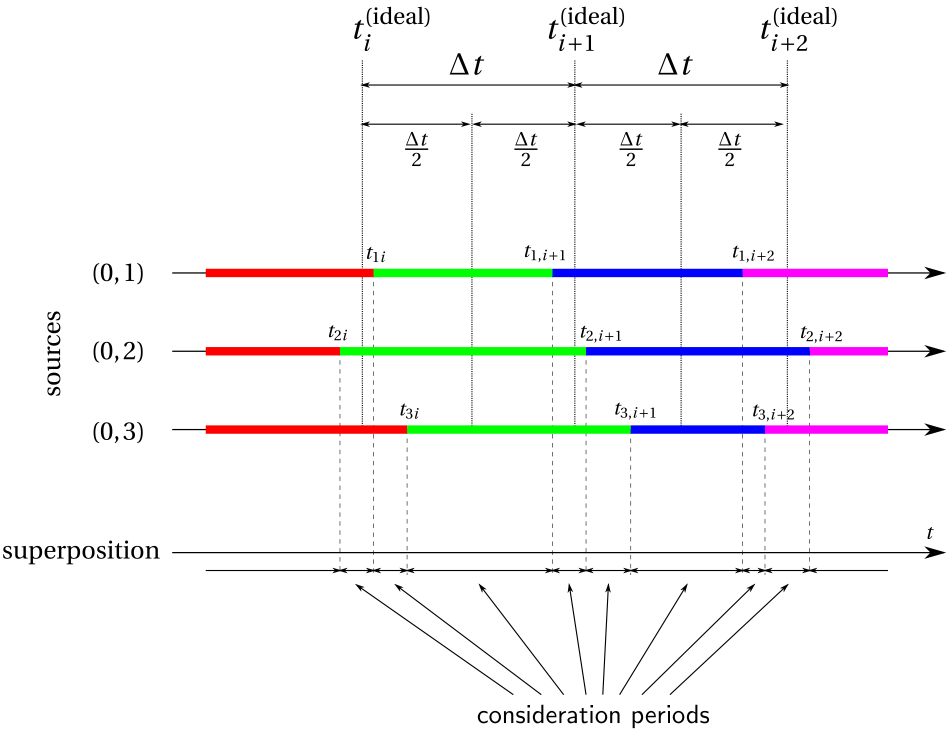


Figure 4.2: Practical scenario with at most two distinct time period IDs in each consideration period (denoted by different colours on timeline of respective source streams)

Figure 4.2 depicts a toy example with three sources conforming to the practical scenario. Data streams emitted by the sources are graphically shown as colour strips on the respective timeline where different colours denote distinct time period IDs. Consideration periods and ideal key-change moments are also indicated in figure 4.2.

A reasonable **condition** for such a practical scenario is to ensure a high enough probability (i.e. confidence level) that all independent random offsets $\{\Omega_k \mid k \in \{1..n_0\}\}$ fall within $\left[-\frac{\Delta t}{2}, \frac{\Delta t}{2}\right]$. In other words, we have the following condition for variances $\{\sigma_k^2\}$ of $\{\Omega_k\}$:

$$\Pr\left[\Omega_k \in \left[-\frac{\Delta t}{2}, \frac{\Delta t}{2}\right], \forall k \in \{1..n_0\}\right] = \prod_{k=1}^{n_0} \int_{-\frac{\Delta t}{2}}^{\frac{\Delta t}{2}} f_{\mathsf{Normal}(0,\sigma_k^2)}(x)\,\mathrm{d}x \geqslant P \tag{4.1}$$

where $f_{\mathsf{Normal}(0,\sigma_k^2)}(x) = \frac{1}{\sigma_k\sqrt{2\pi}} \mathrm{e}^{\frac{-x^2}{2\sigma_k^2}}$ is the Gaussian probability density function of $\mathsf{Normal}\left(0,\sigma_k^2\right)$, and $P \in [0,1]$ is the prescribed confidence level (typically $P \approx 1$).

As long as all variances $\{\sigma_k^2\}$ satisfy equation (4.1), the practical scenario described above applies, implying that it is 'highly unlikely' to have more than two distinct time period IDs in a particular consideration period. This likelihood is so significantly negligible that for all practical purposes, one can reasonably assume that indeed there *are* at most two distinct time period IDs in each consideration period.

4.4 First aggregation layer

This section discusses the rate transfer (from input to output) occurring at processor $(1, m)$ $(\forall m \in \{1..n_1\})$ [definition 2.4] in the first aggregation layer of our parallelisation [definition 2.1].

Lemma 4.1 (Input characteristics at first-layer processor). *In the context of multi-layer parallelisation [definition 2.1], at any time* $t \in \mathbb{R}$, *processor* $(1, m)$ $(m \in \{1..n_1\})$ *receives from sources* $\{(0, k) \mid k \in \{1..n_0\}\}$ *a superposition [definition 3.14] of* n_0 *Poisson sequences* $\{\langle u_{ki}\rangle_i \sim \mathfrak{P}(\lambda_{0:1}) \mid k \in \{1..n_0\}\}$ *[definition 3.13] where mean arrival rate is constant:*

$$\lambda_{0k:1m}(t) = \lambda_{0:1} = \frac{\lambda_0}{n_1} \quad (\forall k \in \{1..n_0\}, \forall m \in \{1..n_1\}, \forall t \in \mathbb{R}) \tag{4.2}$$

Furthermore, within a particular consideration period [definition 4.1] and under the practical-scenario assumption [section 4.3], every $\langle u_{ki}\rangle_i$ *is a constant sequence [definition 3.7] with value set* $\mathcal{V}_k$ *such that* $\mathcal{V} = \bigcup_{k \in \{1..n_0\}} \mathcal{V}_k$ *has cardinality* $|\mathcal{V}| \in \{1, 2\}$.

Proof. Each source $(0, k)$ emits a Poisson sequence with mean arrival rate λ_0 [definition 2.2]. As this is uniformly spread among all processors in layer 1 and data items are routed to $(1, m)$ with probability $p = \frac{1}{n_1}$ [definition 2.2], $(1, m)$ receives a Poisson sequence with rate $p\lambda_0 = \frac{\lambda_0}{n_1}$ from source $(0, k)$ [39, Colouring Theorem, section 5.1, pp. 53–55]. Considering all sources, $(1, m)$ receives a superposition of n_0 Poisson sequences with rate $\frac{\lambda_0}{n_1}$ each.

Also, within a particular consideration period, $(1, m)$ receives from $(0, k)$ a p-thinned copy [definition 3.5] of a constant Poisson sequence [remark 4.1] which remains the same constant Poisson sequence [remark 3.7]. The practical scenario [section 4.3] guarantees that there are at most two distinct time period IDs within a consideration period. Therefore, it follows that every $\langle u_{ki}\rangle_i$ is a constant sequence and $|\mathcal{V}| \in \{1, 2\}$. □

Theorem 4.2 (Total incoming rate at first-layer processor). *In the context of multi-layer parallelisation [definition 2.1] under the practical-scenario assumption [section 4.3], the total incoming rate at processor* $(1, m)$ $(m \in \{1..n_1\})$ *is constant:*

$$\lambda_{1m}^{(in)}(t) = \lambda_1^{(in)} = \frac{n_0\lambda_0}{n_1} \quad (\forall m \in \{1..n_1\}, \forall t \in \mathbb{R}) \tag{4.3}$$

Proof. Processor $(1, m)$ receives a superposition of n_0 Poisson processes $\left\{\mathfrak{P}\left(\frac{\lambda_0}{n_1}\right)\right\}$ [lemma 4.1], hence equation (4.3) [39, Superposition Theorem, section 2.2, pp. 14–17]. □

THEOREM 4.3 (Output characteristics at first-layer processor). *In the context of multi-layer parallelisation [definition 2.1], within a particular consideration period T_c [definition 4.1] and under the practical-scenario assumption [section 4.3], processor $(1, m)$ ($m \in \{1..n_1\}$) outputs Poisson sequence $\langle \tilde{s}_j \rangle \sim \mathfrak{P}\left(\lambda_1^{(out)}\right)$ [definition 3.13] where $\langle \tilde{s}_j \rangle$ is an alternating binary sequence [definition 3.10] (in the long run) and the total outgoing rate is constant:*

$$\lambda_{1m}^{(out)}(t) = \lambda_1^{(out)} = \frac{2n_{0v}(n_0 - n_{0v})\lambda_0}{n_0 n_1} \quad (\forall m \in \{1..n_1\}, \forall t \in T_c) \tag{4.4}$$

where $n_{0v} = |\{k \in \{1..n_0\} \mid \mathcal{V}_k \ni v\}| \in \{1..n_0\}, \forall v \in \mathcal{V}$ (with $\mathcal{V}$ defined in lemma 4.1).

Proof. We have $|\mathcal{V}| \in \{1, 2\}$ [lemma 4.1]. When $|\mathcal{V}| = 1$, there is no output from $(1, m)$ [algorithm 2.1]. In this case, $\lambda_{1m}^{(\text{out})}(t) = 0, \forall m, t$. Equation (4.4) trivially holds.

When $|\mathcal{V}| = 2$, there is a superposition of n_0 constant Poisson sequences each with same rate $\frac{\lambda_0}{n_1}$ at $(1, m)$ [lemma 4.1] resulting in the superposition being a binary sequence. As outputs are triggered at different-consecutive-elements events [algorithm 2.1], the total outgoing rate is $\frac{2n_{0v}(n_0 - n_{0v})}{n_0} \times \frac{\lambda_0}{n_1}$ [corollary 3.11].

Also, algorithm 2.1 implies that output sequence $\langle \tilde{s}_j \rangle$ is the compression [definition 3.4] of the incoming superposition (which is a binary sequence when $|\mathcal{V}| = 2$). Therefore $\langle \tilde{s}_j \rangle$ is an alternating binary sequence [corollary 3.4]. This 'alternating binary' property applies in the long run, ignoring at most one initial element 'drifted' from the previous cluster of key-change moments due to the emission behaviour of algorithm 2.1. □

COROLLARY 4.4 (Upper bound of total outgoing rate at first-layer processor). *In the context of multi-layer parallelisation [definition 2.1] under the practical-scenario assumption [section 4.3], the total outgoing rate of the output Poisson sequence at processor $(1, m)$ ($m \in \{1..n_1\}$) has the following upper bound:*

$$\lambda_{1m}^{(out)}(t) \leqslant \lambda_{1,\max}^{(out)} = \frac{n_0 \lambda_0}{2n_1} \quad (\forall m \in \{1..n_1\}, \forall t \in \mathbb{R}) \tag{4.5}$$

Proof. We have $|\mathcal{V}| \in \{1, 2\}$ [lemma 4.1]. When $|\mathcal{V}| = 1$, $\lambda_{1m}^{(\text{out})}(t) = 0, \forall m, t$ [theorem 4.3]. Equation (4.5) trivially holds. When $|\mathcal{V}| = 2$, there is a superposition of n_0 constant Poisson sequences each with same rate $\frac{\lambda_0}{n_1}$ at $(1, m)$ [lemma 4.1] resulting in a binary superposition sequence. Also, outputs are triggered at different-consecutive-elements events [algorithm 2.1]. Thus the total outgoing rate has upper bound $\frac{n_0}{2} \times \frac{\lambda_0}{n_1}$ [theorem 3.12 and remark 3.23]. □

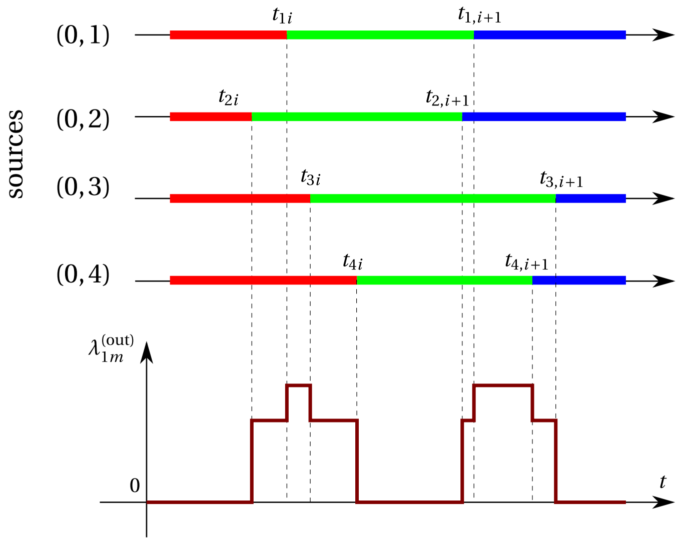


Figure 4.3: Total outgoing rate at first-layer processor as spikes at key-change-moment cluster (colours on timeline denote distinct time period IDs in source streams; diagram not to scale)

With constant $\lambda_{1m}^{\text{(out)}}(t)$ within each consideration period (characterised by $n_{0v} \approx$ the 'mix' ratio between two time period IDs) [theorem 4.3], $\lambda_{1m}^{\text{(out)}}(t)$ across consideration periods is piecewise constant with 'spikes' at key-change-moment clusters. Figure 4.3 shows an illustrative sketch of $\lambda_{1m}^{\text{(out)}}(t)$ in a toy example. The spikes roughly correspond to the discretisation of $f(x)$ in the proof of theorem 3.12 and their peaks correspond to $n_{0v} \approx \frac{n_0}{2}$ (i.e. $\approx 1:1$ mix), which is consistent with theorem 3.12 and corollary 4.4.

4.5 Subsequent aggregation layers

This section discusses the rate transfer (from input to output) occurring at aggregation node (ℓ, m) (processor or sink [definition 2.3], $\forall \ell \in \{2..L\}$, $m \in \{1..n_\ell\}$) in one of the non-first layers in our parallelisation [definition 2.1]. All properties at layer ℓ are recursively dependent on those of the precedent layer $\ell - 1$.

Lemma 4.5 (Input characteristics at non-first-layer aggregation node). *In the context of multi-layer parallelisation [definition 2.1], within a particular consideration period T_c [definition 4.1] and under the practical-scenario assumption [section 4.3], aggregation node (ℓ, m) ($\ell \in \{2..L\}$, $m \in \{1..n_\ell\}$) receives from $\{(\ell-1, k) \mid k \in \{1..n_{\ell-1}\}\}$ a superposition [definition 3.14] of $n_{\ell-1}$ Poisson sequences $\left\{\langle u_{ki}\rangle_i \sim \mathfrak{P}(\lambda_{\ell-1:\ell}) \mid k \in \{1..n_{\ell-1}\}\right\}$ [definition 3.13] where each $\langle u_{ki}\rangle_i$ is an $\frac{1}{n_\ell}$-thinned copy of an unbiased alternating binary sequence [definition 3.10] (in the long run) and its mean arrival rate is constant:*

$$\lambda_{(\ell-1)k:\ell m}(t) = \lambda_{\ell-1:\ell} = \frac{\lambda_{\ell-1}^{(out)}}{n_\ell} \quad (\forall \ell \in \{2..L\}, \forall k \in \{1..n_{\ell-1}\}, \forall m \in \{1..n_\ell\}, \forall t \in T_c) \tag{4.6}$$

Proof. We first consider $\ell = 2$. During T_c, each $(\ell - 1, k) = (1, k)$ emits an alternating binary Poisson sequence with constant mean arrival rate $\lambda^{(\text{out})}_{\ell-1} = \lambda^{(\text{out})}_{1}$ [theorem 4.3]. As this is uniformly spread among all nodes in layer ℓ and data items are routed to (ℓ, m) with probability $p = \dfrac{1}{n_\ell}$ [definition 2.4], (ℓ, m) receives a Poisson sequence with rate $p\lambda^{(\text{out})}_{\ell-1} = \dfrac{\lambda^{(\text{out})}_{\ell-1}}{n_\ell}$ from $(\ell - 1, k)$ [39, Colouring Theorem, section 5.1, pp. 53–55] whose content $\langle u_{ki}\rangle_i$ is a p-thinned copy [definition 3.5] of the above alternating binary Poisson sequence.

Considering the whole layer $\ell - 1$, (ℓ, m) receives a superposition of $n_{\ell-1}$ Poisson sequences, each of which is a p-thinned copy of an alternating binary sequence and has rate $\dfrac{\lambda^{(\text{out})}_{\ell-1}}{n_\ell}$ each. The underlying sequence is equivalently unbiased thanks to the uniform spread.

Repeating this proof recursively by applying theorem 4.7 and lemma 4.5, the conclusion can be obtained $\forall \ell \in \{2..L\}$. Just as theorem 4.3, the 'alternating binary' property applies in the long run, ignoring at most ℓ initial elements 'drifted' from the previous cluster of key-change moments due to the emission behaviour of algorithm 2.1. □

THEOREM 4.6 (Total incoming rate at non-first-layer aggregation node). *In the context of multi-layer parallelisation [definition 2.1] under the practical-scenario assumption [section 4.3], the total incoming rate at aggregation node (ℓ, m) ($\ell \in \{2..L\}, m \in \{1..n_\ell\}$) is constant:*

$$\lambda^{(in)}_{\ell m}(t) = \lambda^{(in)}_{\ell} = \frac{n_{\ell-1}}{n_\ell}\lambda^{(out)}_{\ell-1} \quad (\forall m \in \{1..n_1\}, \forall t \in \mathbb{R}) \tag{4.7}$$

Proof. Node (ℓ, m) receives a superposition of $n_{\ell-1}$ Poisson processes $\left\{\mathfrak{P}\left(\dfrac{\lambda^{(\text{out})}_{\ell-1}}{n_\ell}\right)\right\}$ [lemma 4.5], hence equation (4.7) [39, Superposition Theorem, section 2.2, pp. 14–17]. □

THEOREM 4.7 (Output characteristics at non-first-layer aggregation node). *In the context of multi-layer parallelisation [definition 2.1], within a particular consideration period T_c [definition 4.1] and under the practical-scenario assumption [section 4.3], aggregation node (ℓ, m) ($\ell \in \{2..L\}, m \in \{1..n_\ell\}$) outputs Poisson sequence $\langle \tilde{s}_j\rangle \sim \mathfrak{P}\left(\lambda^{(out)}_{\ell}\right)$ [definition 3.13] where $\langle \tilde{s}_j\rangle$ is an alternating binary sequence [definition 3.10] (in the long run) and the total outgoing rate is constant:*

$$\lambda^{(out)}_{\ell m}(t) = \lambda^{(out)}_{\ell} = \frac{2n_\ell n_{(\ell-1)} - n_{(\ell-1)} + 1}{2n_\ell(2n_\ell - 1)}\lambda^{(out)}_{\ell-1} \quad (\forall \ell \in \{2..L\}, \forall m \in \{1..n_\ell\}, \forall t \in T_c) \tag{4.8}$$

Proof. (ℓ, m) receives a superposition of $n_{\ell-1}$ Poisson sequences, each of which is a $\dfrac{1}{n_\ell}$-

thinned copy of an unbiased alternating binary sequence and has the same mean arrival rate $\frac{\lambda^{(\text{out})}_{\ell-1}}{n_\ell}$ [lemma 4.5]. As outputs are triggered at different-consecutive-elements events [algorithm 2.1], the total outgoing rate is as follows [corollary 3.15]:

$$\lambda^{(\text{out})}_{\ell} = \frac{1}{2}\left(n_{\ell-1} + \frac{\frac{1}{n_\ell}}{2-\frac{1}{n_\ell}}\right)\frac{\lambda^{(\text{out})}_{\ell-1}}{n_\ell} = \frac{2n_\ell n_{(\ell-1)} - n_{(\ell-1)} + 1}{2n_\ell(2n_\ell - 1)}\lambda^{(\text{out})}_{\ell-1}$$

Also, algorithm 2.1 implies that output sequence $\langle \tilde{s}_j \rangle$ is the compression [definition 3.4] of the incoming superposition (which is a binary sequence [remark 3.17]). Therefore $\langle \tilde{s}_j \rangle$ is an alternating binary sequence [corollary 3.4].

Just as lemma 4.5, the 'alternating binary' property applies in the long run, ignoring at most ℓ initial elements 'drifted' from the previous cluster of key-change moments due to the emission behaviour of algorithm 2.1. □

Corollary 4.8 (Upper bound of total outgoing rate at non-first-layer aggregation node). *In the context of multi-layer parallelisation [definition 2.1] under the practical-scenario assumption [section 4.3], the total outgoing rate of the output Poisson sequence at aggregation node (ℓ, m) $(\ell \in \{2..L\}, m \in \{1..n_1\})$ has the following upper bound:*

$$\lambda^{(out)}_{\ell m}(t) \leqslant \lambda^{(out)}_{\ell,\max} = \frac{2n_\ell n_{(\ell-1)} - n_{(\ell-1)} + 1}{2n_\ell(2n_\ell - 1)}\lambda^{(out)}_{(\ell-1),\max} \quad (\forall \ell \in \{2..L\}, \forall m \in \{1..n_1\}, \forall t \in \mathbb{R}) \tag{4.9}$$

Proof. Node (ℓ, m) receives a superposition of $n_{\ell-1}$ Poisson sequences each of which is a $\frac{1}{n_\ell}$-thinned copy of an unbiased alternating binary sequence and has the same mean arrival rate $\frac{\lambda^{(\text{out})}_{\ell-1}}{n_\ell}$ [lemma 4.5].

Also, we have $\frac{\lambda^{(\text{out})}_{\ell-1}}{n_\ell} \in \left[0, \frac{\lambda^{(\text{out})}_{(\ell-1),\max}}{n_\ell}\right]$ [corollary 4.8 recursively] and outputs are triggered at different-consecutive-elements events [algorithm 2.1]. Thus the total outgoing rate has the following upper bound [theorem 3.16]:

$$\lambda^{(\text{out})}_{\ell,\max} = \frac{1}{2}\left(n_{\ell-1} + \frac{\frac{1}{n_\ell}}{2-\frac{1}{n_\ell}}\right)\frac{\lambda^{(\text{out})}_{(\ell-1),\max}}{n_\ell} = \frac{2n_\ell n_{(\ell-1)} - n_{(\ell-1)} + 1}{2n_\ell(2n_\ell - 1)}\lambda^{(\text{out})}_{(\ell-1),\max}$$

□

4.6 Consolidation for all layers

THEOREM 4.9 (Upper bound of total incoming rate at aggregation node). *In the context of multi-layer parallelisation [definition 2.1] under the practical-scenario assumption [section 4.3], total incoming rate at any aggregation node* (ℓ, m) *(*$\ell \in \{1..L\}, m \in \{1..n_1\}$*) has the following upper bound:*

$$\lambda^{(in)}_{\ell m}(t) \leqslant \lambda^{(in)}_{\ell,\max} = \frac{n_0 \lambda_0}{2^{\ell-1} n_\ell} \prod_{k=1}^{\ell-2} \frac{2 n_{k+1} n_k - n_k + 1}{n_k (2 n_{k+1} - 1)} \quad (\forall \ell \in \{1..L\}, \forall m \in \{1..n_\ell\}, \forall t \in \mathbb{R}) \tag{4.10}$$

Proof. Intensively applying various results from sections 4.4 and 4.5 yields:

$$\begin{aligned}
\lambda^{(\text{in})}_{1,\max} &= \lambda^{(\text{in})}_{1} = \frac{n_0 \lambda_0}{n_1} \quad \text{[theorem 4.2]} \\
\lambda^{(\text{out})}_{1,\max} &= \frac{n_0 \lambda_0}{2 n_1} \quad \text{[corollary 4.4]}
\end{aligned} \tag{4.11}$$

$$\begin{aligned}
\lambda^{(\text{in})}_{2,\max} &= \frac{n_1}{n_2} \lambda^{(\text{out})}_{1,\max} = \frac{n_1}{n_2} \times \frac{n_0 \lambda_0}{2 n_1} = \frac{n_0 \lambda_0}{2 n_2} \quad \text{[theorem 4.6 and equation (4.11)]} \\
\lambda^{(\text{out})}_{2,\max} &= \frac{2 n_2 n_1 - n_1 + 1}{2 n_2 (2 n_2 - 1)} \lambda^{(\text{out})}_{1,\max} = \frac{2 n_2 n_1 - n_1 + 1}{2 n_2 (2 n_2 - 1)} \times \frac{n_0 \lambda_0}{2 n_1} \\
&= \frac{n_0 \lambda_0 (2 n_2 n_1 - n_1 + 1)}{2^2 n_1 n_2 (2 n_2 - 1)} \quad \text{[corollary 4.8 and equation (4.11)]}
\end{aligned} \tag{4.12}$$

$$\begin{aligned}
\lambda^{(\text{in})}_{3,\max} &= \frac{n_2}{n_3} \lambda^{(\text{out})}_{2,\max} = \frac{n_2}{n_3} \times \frac{n_0 \lambda_0 (2 n_2 n_1 - n_1 + 1)}{2^2 n_1 n_2 (2 n_2 - 1)} \\
&= \frac{n_0 \lambda_0 (2 n_2 n_1 - n_1 + 1)}{2^2 n_3 (n_1) (2 n_2 - 1)} \quad \text{[theorem 4.6 and equation (4.12)]} \\
\lambda^{(\text{out})}_{3,\max} &= \frac{2 n_3 n_2 - n_2 + 1}{2 n_3 (2 n_3 - 1)} \lambda^{(\text{out})}_{2,\max} = \frac{2 n_3 n_2 - n_2 + 1}{2 n_3 (2 n_3 - 1)} \times \frac{n_0 \lambda_0 (2 n_2 n_1 - n_1 + 1)}{2^2 n_1 n_2 (2 n_2 - 1)} \\
&= \frac{n_0 \lambda_0 (2 n_3 n_2 - n_2 + 1)(2 n_2 n_1 - n_1 + 1)}{2^3 n_1 n_2 n_3 (2 n_3 - 1)(2 n_2 - 1)} \quad \text{[corollary 4.8 and equation (4.12)]}
\end{aligned} \tag{4.13}$$

$$\begin{aligned}
\lambda^{(\text{in})}_{4,\max} &= \frac{n_3}{n_4} \lambda^{(\text{out})}_{3,\max} = \frac{n_3}{n_4} \times \frac{n_0 \lambda_0 (2 n_3 n_2 - n_2 + 1)(2 n_2 n_1 - n_1 + 1)}{2^3 n_1 n_2 n_3 (2 n_3 - 1)(2 n_2 - 1)} \\
&= \frac{n_0 \lambda_0 (2 n_3 n_2 - n_2 + 1)(2 n_2 n_1 - n_1 + 1)}{2^3 n_4 (n_1 n_2)(2 n_3 - 1)(2 n_2 - 1)} \quad \text{[theorem 4.6 and equation (4.13)]} \\
\lambda^{(\text{out})}_{4,\max} &= \dots
\end{aligned}$$

$$\therefore \lambda^{(\text{in})}_{\ell,\max} = \frac{n_0 \lambda_0}{2^{\ell-1} n_\ell} \prod_{k=1}^{\ell-2} \frac{2 n_{k+1} n_k - n_k + 1}{n_k (2 n_{k+1} - 1)} \quad (\forall \ell \in \{1..L\})$$

□

5 Topology determination

With a closed-form formula for the upper bound of total incoming rate at any aggregation node $\lambda_{\ell,\max}^{(\text{in})}$ $(\forall \ell \in \{1..L\})$ [theorem 4.9], we are ready to revisit the problem statement in section 5.1 and explore systematic approaches to topology determination.

Considering various objectives and circumstances, the problem statement [section 2.2] can be transformed in different ways for practical solutions. Some of these could be slightly more restrictive than others with respect to the original problem. However, all are valid and plausible, as outlined in section 5.2 and discussed in sections 5.3 to 5.6.

5.1 Problem statement revisited

In this section, we revisit the formalised problem statement [section 2.2] for topology determination. Applying theorem 4.9, the ingest rate constraint of definition 2.7 becomes:

$$\lambda_{\ell,\max}^{(\text{in})} \leqslant \theta \Leftrightarrow \psi_\ell(n_1, n_2, \ldots, n_\ell) \leqslant 0 \quad (\forall \ell \in \{1..L\}) \tag{5.1}$$

where function $\psi_\ell(n_1, n_2, \ldots, n_\ell)$ is defined as follows $\forall \ell \in \{1..L\}$:

$$\psi_\ell(n_1, n_2, \ldots, n_\ell) = n_0\lambda_0 \prod_{k=1}^{\ell-2} (2n_{k+1}n_k - n_k + 1) - \theta 2^{\ell-1} n_\ell \prod_{k=1}^{\ell-2} n_k(2n_{k+1} - 1) \tag{5.2}$$

Using equation (5.1) above, the formalised problem statement [equation (2.2), section 2.2] can then be restated more precisely as:

$$\begin{cases} \min\limits_{n_1, n_2, \ldots, n_{L-1}} \sum\limits_{\ell=1}^{L-1} n_\ell \\ \min\limits_{L} L \\ \text{subject to: } L, n_1, n_2, \ldots, n_{L-1} \in \mathbb{N}^* \\ \text{subject to: } \psi_\ell(n_1, n_2, \ldots, n_\ell) \leqslant 0 \quad (\forall \ell \in \{1..L\}) \end{cases} \tag{5.3}$$

The problem presented in equation (5.3) is actually *ill-posed* as there are two 'independent' minimisations (i.e. two objective functions) without a clear indication of their relative priority. It will no longer be ill-posed if we introduce parameters $\beta, \gamma \in \mathbb{R}_+^*$ to linearly combine the two:

$$\begin{cases} \min\limits_{L, n_1, n_2, \ldots, n_{L-1}} \left(\beta \sum\limits_{\ell=1}^{L-1} n_\ell + \gamma L \right) \\ \text{subject to: } L, n_1, n_2, \ldots, n_{L-1} \in \mathbb{N}^* \\ \text{subject to: } \psi_\ell(n_1, n_2, \ldots, n_\ell) \leqslant 0 \quad (\forall \ell \in \{1..L\}) \end{cases} \tag{5.4}$$

Certainly, parameters β and γ ought to be specified a priori. Although this new problem is not precisely the original ill-posed one, it is a reasonable approximation.

NB. While there is always one sink, i.e. constant $n_L = 1$ [definition 2.1], it is convenient at times (especially in solution implementation) to incorporate n_L as an extra variable together with a trivial linear range constraint. For instance, on top of equation (5.4), we have:

$$\begin{cases} \min\limits_{L, n_1, n_2, \ldots, n_L} \left(\beta \sum\limits_{\ell=1}^{L} n_\ell + \gamma L \right) \\ \text{subject to: } L, n_1, n_2, \ldots, n_{L-1} \in \mathbb{N}^* \\ \text{subject to: } n_L = 1 \\ \text{subject to: } \psi_\ell(n_1, n_2, \ldots, n_\ell) \leqslant 0 \quad (\forall \ell \in \{1..L\}) \end{cases} \tag{5.5}$$

5.2 Road map to topology determination

As can be seen from section 5.1, topology determination boils down to finding a plausible assignment for $L, n_1, n_2, \ldots, n_{L-1}$ according to the requirements stated in equation (5.4). In fact, the well-posed problem in equation (5.4) resembles **mathematical optimisation** with L integer variables ($L, n_1, n_2, \ldots, n_{L-1}$), L linear range constraints (all variables being positive), L non-linear inequality constraints (equation (5.1)) and a linear objective function.

Nevertheless, such optimisation is rather **non-standard** owing to the presence of optimisation variable L in the indices. Specifically, the number of variables and constraints are *not fixed*. They are all dependent on L, *itself* an optimisation variable, while mathematical optimisation generally operates on a prescribed number of variables which need to satisfy a fixed number of constraints while optimising the objective function.

Consequently, some relaxation is necessary to **transform** either the ill-posed [equation (5.3)] or non-standard problem [equation (5.4)] so that plausible solutions can be reached and systematic approaches to topology determination can be identified. Any relaxation will slightly alter the original scenario; however, as the reader will soon discover, the transforms employed in this work are admissible under practical circumstances.

Our road map to topology determination is depicted in figure 5.1. Various alternative solution paths will be presented in subsequent sections, as outlined below. These constitute systematic approaches to multi-layer parallelisation of time-based stream aggregation, based on solid analytical foundation [chapters 3 and 4] and in accordance with the methodology adopted [section 2.3].

- To tackle the **original ill-posed** problem [equation (5.3)], we can *prioritise* on either of the two minimisations or employ *heuristics* specific to the constraints [equation (5.1)]. This leads to brute-force approaches [section 5.3], mixed-integer non-linear programming (MINLP) [section 5.4] or conservative sequential assignments [section 5.5].
- To tackle the **well-posed non-standard** problem [equation (5.4)], we can cap L at some $L_{\max} \in \mathbb{N}^*$. This also leads to a MINLP-based approach [section 5.6].

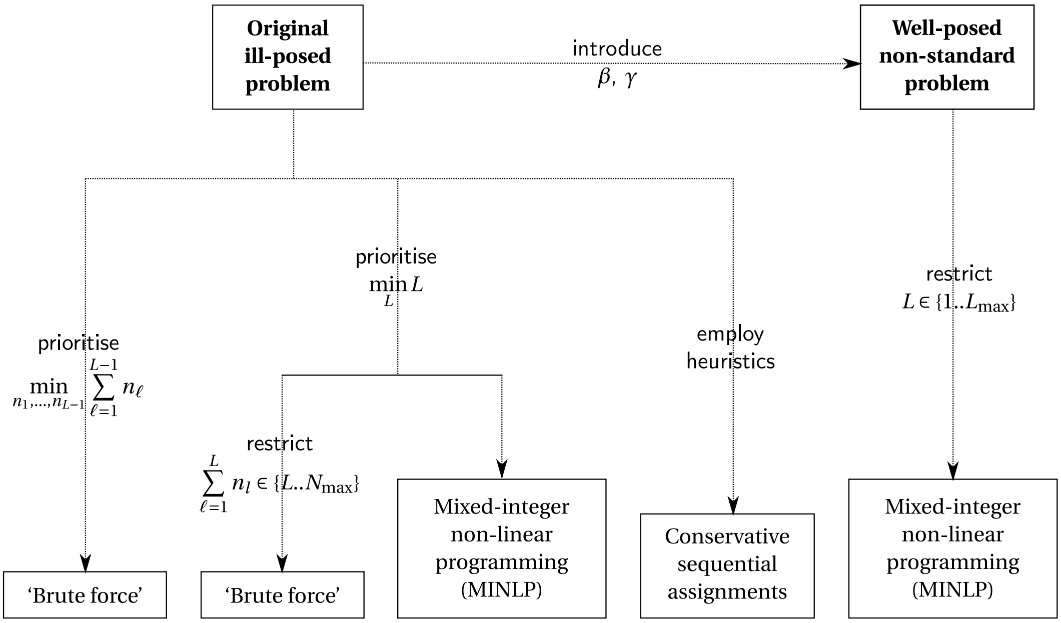

Original ill-posed problem
introduce β, γ
Well-posed non-standard problem
prioritise $\min_{n_1,\dots,n_{L-1}} \sum_{\ell=1}^{L-1} n_\ell$
prioritise $\min_L L$
restrict $\sum_{\ell=1}^{L} n_l \in \{L..N_{\max}\}$
employ heuristics
restrict $L \in \{1..L_{\max}\}$
'Brute force'
'Brute force'
Mixed-integer non-linear programming (MINLP)
Conservative sequential assignments
Mixed-integer non-linear programming (MINLP)

Figure 5.1: Road map to topology determination

It is noteworthy that these approaches are in stark contrast with over-provision or trial-and-error. Overly large n_ℓ and L may satisfy all constraints [equation (5.1)] but could result in costly and inefficient set-ups (cf. sections 1.6 and 2.2). The reader is also reminded that the worst-case scenario has been adopted in the derivation [chapters 3 and 4] so that all solutions ensure non-saturation of ingest rate constraint [definition 2.7].

5.3 Priority-based brute force

Brute force or *exhaustive search* is never a good idea as it is not scalable. Nonetheless, it is a good starting point before venturing into better solutions. Brute-force approaches offer an opportunity to examine potential solution sketches, especially for ill-posed problems.

In the ill-posed problem of equation (5.3), both objective functions map to $\mathbb{N}^*$. For *each objective alone* (assuming the other being constant), brute-force minimisation can proceed as follows: We first assign 1 as the objective function value, derive all possible variable assignments that result in such value and check these against the constraints. If an assignment satisfies all constraints, it is deemed the 'optimal' solution and the process returns. If all fail, we increment the objective function value by 1 then repeat until a solution is found or some stop condition applies (e.g. time-out, cap on objective function value, etc.)

When *both objective functions* are in effect, we can *prioritise* either of them so as to remain brute-force manner without the linear combination of equation (5.4). Prioritisation refers to the same conservative increments of individual objective function values as above. However, these should now proceed in two nested loops, the outer of which is for the prioritised function. We propose this method as the **priority-based brute-force approach** to topology determination as algorithms 5.1 and 5.2 which prioritise the minimisation of the *number of aggregation nodes* $N = \sum_{\ell=1}^{L} n_\ell$ and the *number of layers* L, respectively.

Algorithm 5.1: Brute-force topology determination: prioritise $\min_{n_1,\ldots,n_{L-1}} \sum_{\ell=1}^{L-1} n_\ell$

```
Input: $n_0 \in \mathbb{N}^*, \lambda_0 \in \mathbb{R}^+, \theta \in \mathbb{R}^+, n_L = 1$
Output: $L, n_1, n_2, \ldots, n_{L-1} \in \mathbb{N}^*$
if $\psi_1(1) \leqslant 0$ then $L \leftarrow 1$; return $L$; ;
for $N \leftarrow 2$ to $+\infty$ do
    for $L \leftarrow 2$ to $N$ do
        $\mathscr{P} = \left\{\left(n_1^{(i)}, n_2^{(i)}, \ldots, n_{L-1}^{(i)}\right) \middle| i \in \{1..|\mathscr{P}|\}\right\} \leftarrow$ ordered-partition($N-1, L-1$);
        for $i \leftarrow 1$ to $|\mathscr{P}|$ do
            $n_L^{(i)} \leftarrow 1$;
            if $\psi_\ell\left(n_1^{(i)}, n_2^{(i)}, \ldots, n_\ell^{(i)}\right) \leqslant 0, \forall \ell \in \{1..L\}$ then
                $n_1 \leftarrow n_1^{(i)}$; $n_2 \leftarrow n_2^{(i)}$; ...; $n_{L-1} \leftarrow n_{L-1}^{(i)}$;    return $L, n_1, n_2, \ldots, n_{L-1}$;
```

In **algorithm 5.1**, no *stop condition* is shown for the sake of brevity. When such condition applies but solutions have yet to be found, the algorithm returns `null` indicating 'no feasible solution'. Moreover, $n_\ell \geqslant 1, \forall \ell \in \{1..L\} \Leftrightarrow N = \sum_{\ell=1}^{L} n_\ell \geqslant L$. Therefore, the inner loop is *restricted* to $L \leqslant N$, ensuring that each outer-loop iteration never runs infinitely. Also, the *special case* $N = L = 1$ is treated separately outside the nested loops. Moving to **algorithm 5.2**, there is no major difference except the *swapping* of the loops and the existence of an $N_{\max}$ cap to prevent outer-loop iterations from running infinitely. As shown above, $L \leqslant N \leqslant N_{\max}$ so the outer loop is also restricted to $L \leqslant N_{\max}$ for consistency. Consequently, even without another stop condition, the algorithm may still exhaust all possibilities and return `null`.

Algorithm 5.2: Brute-force topology determination: prioritise $\min_{L} L$, restrict $\sum_{\ell=1}^{L} n_\ell \in \{L..N_{\max}\}$

```
Input: $n_0 \in \mathbb{N}^*$, $\lambda_0 \in \mathbb{R}^+$, $\theta \in \mathbb{R}^+$, $N_{\max} \in \mathbb{N}^*$, $n_L = 1$
Output: $L, n_1, n_2, \ldots, n_{L-1} \in \mathbb{N}^*$
if $\psi_1(1) \leqslant 0$ then $L \leftarrow 1$; return $L$; ;
if $N_{\max} < 2$ then return null; ;
for $L \leftarrow 2$ to $N_{\max}$ do
    for $N \leftarrow L$ to $N_{\max}$ do
        $\mathscr{P} = \left\{ \left( n_1^{(i)}, n_2^{(i)}, \ldots, n_{L-1}^{(i)} \right) \middle| \, i \in \{1..|\mathscr{P}|\} \right\} \leftarrow$ ordered-partition($N-1, L-1$);
        for $i \leftarrow 1$ to $|\mathscr{P}|$ do
            $n_L^{(i)} \leftarrow 1$;
            if $\psi_\ell \left( n_1^{(i)}, n_2^{(i)}, \ldots, n_\ell^{(i)} \right) \leqslant 0, \forall \ell \in \{1..L\}$ then
                $n_1 \leftarrow n_1^{(i)}$; $n_2 \leftarrow n_2^{(i)}$; $\ldots$; $n_{L-1} \leftarrow n_{L-1}^{(i)}$;    return $L, n_1, n_2, \ldots, n_{L-1}$;
return null;
```

In both algorithms, **subroutine** `ordered-partition`(n, p) returns the set of all possible ordered p-partitions of n, i.e. all ordered partitions of n into a sum of p terms $(n, p \in \mathbb{N}^*)$. A plausible algorithm for `ordered-partition`(n, p) is to line up n 1's then find all possible unordered placements of $p-1$ commas in $n-1$ spaces in between the 1's. For each placement, '+' signs can be inserted into the remaining spaces then simple arithmetic yields the result. As there are $\mathrm{C}_{n-1}^{p-1} = \dfrac{(n-1)!}{(p-1)!(n-p)!}$ placements of commas, there are C_{n-1}^{p-1} such partitions.

***Example* 5.1.** $\mathscr{R} = \{(1,1,3),(1,2,2),(1,3,1),(2,1,2),(2,2,1),(3,1,1)\}$ is the output when invoking `ordered-partition`$(5,3)$. Indeed, we have $|\mathscr{R}| = \mathrm{C}_4^2 = \dfrac{4!}{2!2!} = 6$ elements.

5.4 L-priority mixed-integer non-linear programming

In algorithm 5.3, the inner loop deals with a transformed problem where L is no longer a variable. With constant L, the number of optimisation variables is fixed $(L-1)$, so are the number of linear range constraints $(L-1)$ and non-linear inequality constraints (L). In particular, we have the following problem, for every L dictated by the outer loop of algorithm 5.3.

$$\begin{cases} \min\limits_{n_1, n_2, \ldots, n_{L-1}} \sum\limits_{\ell=1}^{L-1} n_\ell \\ \text{subject to: } n_1, n_2, \ldots, n_{L-1} \in \mathbb{N}^* \\ \text{subject to: } \psi_\ell(n_1, n_2, \ldots, n_\ell) \leqslant 0 \quad (\forall \ell \in \{1..L\}) \end{cases} \tag{5.6}$$

The problem in equation (5.6) is indeed mixed-integer non-linear programming (MINLP) [42, 43] although there are only integer variables and a linear objective function. Inspired by the L-priority approach as in algorithm 5.2 but employing a standard off-the-shelf MINLP solver instead of the naïve brute-force inner-loop, we obtain algorithm 5.3. This is the **L-priority MINLP-based approach** proposed for multi-layer parallelisation.

Algorithm 5.3: MINLP-based topology determination, prioritising $\min_L L$

```
Input: $n_0 \in \mathbb{N}^*$, $\lambda_0 \in \mathbb{R}^+$, $\theta \in \mathbb{R}^+$, $n_L = 1$
Output: $L, n_1, n_2, \ldots, n_{L-1} \in \mathbb{N}^*$
for $L \leftarrow 1$ to $+\infty$ do
    variables $\mathcal{V} \leftarrow \{n_1, n_2, \ldots, n_{L-1}\}$;
    objective $\omega \leftarrow \left[\min_{n_1, n_2, \ldots, n_{L-1}} \sum_{\ell=1}^{L-1} n_\ell\right]$;
    range constraints $\mathcal{C}_1 \leftarrow \{[n_\ell \in \mathbb{N}^*] \mid \ell \in \{1..L-1\}\}$;
    general constraints $\mathcal{C}_2 \leftarrow \{[\psi_\ell(n_1, n_2, \ldots, n_\ell) \leqslant 0] \mid \ell \in \{1..L\}\}$;
    $(n_1, n_2, \ldots, n_{L-1}) \leftarrow$ MINLP-solver($\mathcal{V}$,$\omega$,$\mathcal{C}_1 \bigcup \mathcal{C}_2$);
    if $(n_1, n_2, \ldots, n_{L-1}) \neq null$ then
        return $L, n_1, n_2, \ldots, n_{L-1}$;
```

Just as in section 5.3, no stop condition like time-out or $L_{\max}$ is shown for the sake of conciseness. Variables, objective and constraints supplied as parameters to the MINLP solver closely follow equation (5.6). No $N_{\max}$ cap is required as the solver should be able to handle standard MINLP without additional constraints. Of course, it may implement its own stop condition, resulting in possible 'no feasible solution'.

Regarding **MINLP** in general, the problem has been intensively studied [42, 43] and various techniques have been implemented as off-the-shelf optimisation suites. For instance, an extension of the ant colony optimisation meta-heuristic and the oracle penalty method for constraint handling [44] are employed in MIDACO [45], a MINLP-focused general-purpose optimiser. Similarly, SCIP [46], a mixed-integer programming (MIP) and MINLP solver, integrates constraint programming with MIP modelling and solving techniques [47, 48, 49].

Given the sets of variables $\mathcal{V}$ and constraints $\mathcal{C}$ as well as the objective ω, MIDACO and SCIP **solvers** are feasible candidates to realise subroutine `MINLP-solver(`$\mathcal{V}$,ω,$\mathcal{C}$`)` of algorithm 5.3. It is worth highlighting that MINLP solvers are employed as *black boxes* as the details of their solving techniques are not the subject of this study.

5.5 Conservative sequential assignments

We revisit the constraints [equation (5.1)] which can be equivalently transformed as follows:

$$n_\ell \geqslant \frac{n_0\lambda_0}{\theta 2^{\ell-1}} \prod_{k=1}^{\ell-2} \frac{2n_{k+1}n_k - n_k + 1}{n_k(2n_{k+1}-1)} = \tilde{\psi}_\ell(n_1, n_2, \ldots, n_{\ell-1}) \quad (\forall \ell \in \{1..L\}) \tag{5.7}$$

As can be seen from equation (5.7), the criterion for n_ℓ at layer ℓ relies on the outcome of function $\tilde{\psi}_\ell(n_1, n_2, \ldots, n_{\ell-1})$ which *only* depends on the numbers of nodes at layers *preceding* ℓ. This makes **sequential assignments** of n_ℓ with incremental ℓ possible, resembling the L-priority idea of algorithm 5.2.

The minimisation of $\sum\limits_{\ell=1}^{L} n_\ell$ can then be achieved by **conservatively** selecting the smallest possible value for every n_ℓ. As $n_\ell \geqslant \tilde{\psi}_\ell(n_1, n_2, \ldots, n_{\ell-1})$ [equation (5.7)] and $n_\ell \in \mathbb{N}^*$, this corresponds to picking $n_\ell = \lceil \tilde{\psi}_\ell(n_1, n_2, \ldots, n_{\ell-1}) \rceil$. The process terminates as soon as $n_\ell = 1$ at some ℓ (the single sink [definition 2.5], with ℓ being the finalised number of layers L).

Conservative sequential assignments are proposed in algorithm 5.4 as another systematic approach to multi-layer parallelisation. It does not involve any search but rather employing a heuristic specific to the ill-posed problem [equation (5.3)] based on equation (5.7).

Algorithm 5.4: Conservative sequential assignments for topology determination

```
Input: $n_0 \in \mathbb{N}^*$, $\lambda_0 \in \mathbb{R}^+$, $\theta \in \mathbb{R}^+$, $N_{\max} \in \mathbb{N}^*$
Output: $L, n_1, n_2, \ldots, n_L \in \mathbb{N}^*$
for $\ell \leftarrow 1$ to $+\infty$ do
    $n_\ell \leftarrow \left\lceil \frac{n_0\lambda_0}{\theta 2^{\ell-1}} \prod_{k=1}^{\ell-2} \frac{2n_{k+1}n_k - n_k + 1}{n_k(2n_{k+1}-1)} \right\rceil$;
    if $n_\ell = 1$ then
        $L \leftarrow \ell$;    return $L, n_1, n_2, \ldots, n_L$;
```

The results obtained from algorithm 5.4 may not be optimal in the strict sense, which is not clearly defined anyway for an ill-posed problem like equation (5.3). However, in some practical cases, it could be more efficient than those presented earlier [sections 5.3 and 5.4].

5.6 L-cap mixed-integer non-linear programming

This section attempts to tackle the **well-posed non-standard** problem [equation (5.4)] by capping $L \in \{1..L_{\max}\}$ for some $L_{\max} \in \mathbb{N}^*$. With $L_{\max}$ in place, we can remove the problem's non-standard aspect discussed in [section 5.2] — i.e. making the number of variables and constraints static to properly fit a MINLP formulation.

In particular, dummy optimisation variables n_{L+1} to $n_{(L_{\max})}$ can be introduced alongside n_1 to n_L (NB. $n_L = 1$ incorporated as a variable for convenience here, cf. section 5.1). Dummy variables are distinguished from the rest by $L_{\max}$ binary masks $\{m_\ell \in \{0,1\} \mid \ell \in \{1..L_{\max}\}\}$ where $m_\ell = 0$ indicates that the corresponding n_ℓ is dummy and does not constitute a layer in the resultant topology. With the total number of non-dummy layers $L \in \{1..L_{\max}\}$, we must have the following so that binary masks m_ℓ are valid:

$$m_\ell = \begin{cases} 1 & \text{if } \ell \in \{1..L\} \\ 0 & \text{if } \ell \in \{(L+1)..L_{\max}\} \end{cases} \tag{5.8}$$

In other words, as ℓ increments, binary masks $\{m_\ell\}$ are non-increasing with fixed initial $m_1 = 1$. Index ℓ such that $m_\ell = 1$ and $m_{\ell+1} = 0$ is the total number of non-dummy layers L; we must have $n_L = 1$. Previously problematic variable L (causing dynamic number of variables and constraints) can now be represented by a fixed number of binary masks $\{m_\ell\}$ as additional optimisation variables. The following newly introduced constraints incorporate the above non-increasing property (with dummy fixed $m_{(L_{\max}+1)} = 0$ for convenience) as well as the restriction $n_L = 1$ at the single decreasing point of masks $\{m_\ell\}$.

$$0 \leqslant n_\ell(m_\ell - m_{\ell+1}) \leqslant 1 \quad (\forall \ell \in \{1..L_{\max}\}) \tag{5.9}$$

We also need to adapt the original constraints of equation (5.1) as these do not apply to all variables n_1 to $n_{(L_{\max})}$ but only non-dummy ones n_1 to n_L. This can be achieved simply by multiplying the corresponding binary mask m_ℓ to the ℓ^{th} constraint:

$$\begin{aligned} & \psi_\ell(n_1, n_2, \ldots, n_\ell) \leqslant 0 \quad (\forall \ell \in \{1..L\}) \\ \Leftrightarrow \quad & m_\ell \psi_\ell(n_1, n_2, \ldots, n_\ell) \leqslant 0 \quad (\forall \ell \in \{1..L_{\max}\}) \end{aligned} \tag{5.10}$$

With all constraints in place, the objective function can be adapted in the same manner. Note that fixed $m_1 = 1$ is not considered a variable for the sake of simplicity:

$$\begin{aligned} & \min_{L, n_1, n_2, \ldots, n_L} \left(\beta \sum_{\ell=1}^{L} n_\ell + \gamma L \right) \\ \Leftrightarrow \quad & \min_{\substack{n_1, n_2, \ldots, n_{(L_{\max})} \\ m_2, m_3, \ldots, m_{(L_{\max})}}} \left(\beta \sum_{\ell=1}^{L_{\max}} n_\ell m_\ell + \gamma \sum_{\ell=1}^{L_{\max}} m_\ell \right) \end{aligned} \tag{5.11}$$

Putting everything together, we obtain a MINLP problem [42, 43] similar to section 5.4, which can be solved with an off-the-shelf MINLP solver like MIDACO [50] or SCIP [46]:

$$\begin{cases} \min\limits_{\substack{n_1,n_2,\ldots,n_{(L_{\max})}\\ m_2,m_3,\ldots,m_{(L_{\max})}}} \left(\beta \sum\limits_{\ell=1}^{L_{\max}} n_\ell m_\ell + \gamma \sum\limits_{\ell=1}^{L_{\max}} m_\ell\right) \\ \text{subject to: } n_\ell \in \mathbb{N}^* \quad (\forall \ell \in \{1..L_{\max}\}) \\ \text{subject to: } m_\ell \in \{0,1\} \quad (\forall \ell \in \{2..L_{\max}\}) \\ \text{subject to: } 0 \leqslant n_\ell(m_\ell - m_{\ell+1}) \leqslant 1 \quad (\forall \ell \in \{1..L_{\max}\}) \\ \text{subject to: } m_\ell \psi_\ell(n_1,n_2,\ldots,n_\ell) \leqslant 0 \quad (\forall \ell \in \{1..L_{\max}\}) \end{cases} \tag{5.12}$$

We propose this as an *L*-**cap MINLP-based approach** to multi-layer parallelisation in algorithm 5.5. More variables need to be involved $(2L_{\max} - 1)$ and generally larger $L_{\max}$ is preferred to achieve feasible solutions. As can be seen from algorithm 5.5, the process can be wrapped in an incremental-$L_{\max}$ loop for greater flexibility, avoiding the heuristic prescription of $L_{\max}$.

Algorithm 5.5: MINLP-based topology determination, restricting $L \in \{1..L_{\max}\}$ $(L_{\max} \in \mathbb{N}^*)$

```
Input: $n_0 \in \mathbb{N}^*$, $\lambda_0 \in \mathbb{R}^+$, $\theta \in \mathbb{R}^+$, $L_{\max} \in \mathbb{N}^*$, $m_1 = 1$, $m_{(L_{\max}+1)} = 0$
Output: $L, n_1, n_2, \ldots, n_L \in \mathbb{N}^*$

variables $\mathcal{V} \leftarrow \{n_1, n_2, \ldots, n_{(L_{\max})}\} \cup \{m_2, m_3, \ldots, m_{(L_{\max})}\}$;
objective $\omega \leftarrow \left[\min\limits_{\substack{n_1,n_2,\ldots,n_{(L_{\max})}\\ m_2,m_3,\ldots,m_{(L_{\max})}}} \left(\beta \sum_{\ell=1}^{L_{\max}} n_\ell m_\ell + \gamma \sum_{\ell=1}^{L_{\max}} m_\ell\right)\right]$;
range constraints $\mathcal{C}_1 \leftarrow \{[n_\ell \in \mathbb{N}^*] \mid \ell \in \{1..L_{\max}\})\}$;
$\mathcal{C}_1 \leftarrow \mathcal{C}_1 \cup \{[m_\ell \in \{0,1\}] \mid \ell \in \{2..L_{\max}\}\}$;
general constraints $\mathcal{C}_2 \leftarrow \{[0 \leqslant n_\ell(m_\ell - m_{\ell+1}) \leqslant 1] \mid \ell \in \{1..L_{\max}\}\}$;
$\mathcal{C}_2 \leftarrow \mathcal{C}_2 \cup \{[m_\ell \psi_\ell(n_1, n_2, \ldots, n_\ell) \leqslant 0] \mid \ell \in \{1..L_{\max}\}\}$;
$(n_1, n_2, \ldots, n_{(L_{\max})}, m_2, m_3, \ldots, m_{(L_{\max})}) \leftarrow$ MINLP-solver($\mathcal{V}, \omega, \mathcal{C}_1 \cup \mathcal{C}_2$);
if $(n_1, n_2, \ldots, n_{(L_{\max})}, m_2, m_3, \ldots, m_{(L_{\max})}) = null$ then
    return null;
for $\ell \leftarrow 1$ to $L_{\max}$ do
    if $m_{\ell+1} = 0$ then
        $L \leftarrow \ell$;    break;
return $L, n_1, n_2, \ldots, n_L$;
```

6 Simulations & prototypes

6.1 Empirical verification of corollary 3.10

Besides mathematical analyses [chapter 3], a computer simulation was conducted in this study to empirically verify **corollary 3.10** on the superposition of constant Poisson sequences, one of the major foundations for first-layer rate transfer analyses [section 4.4].

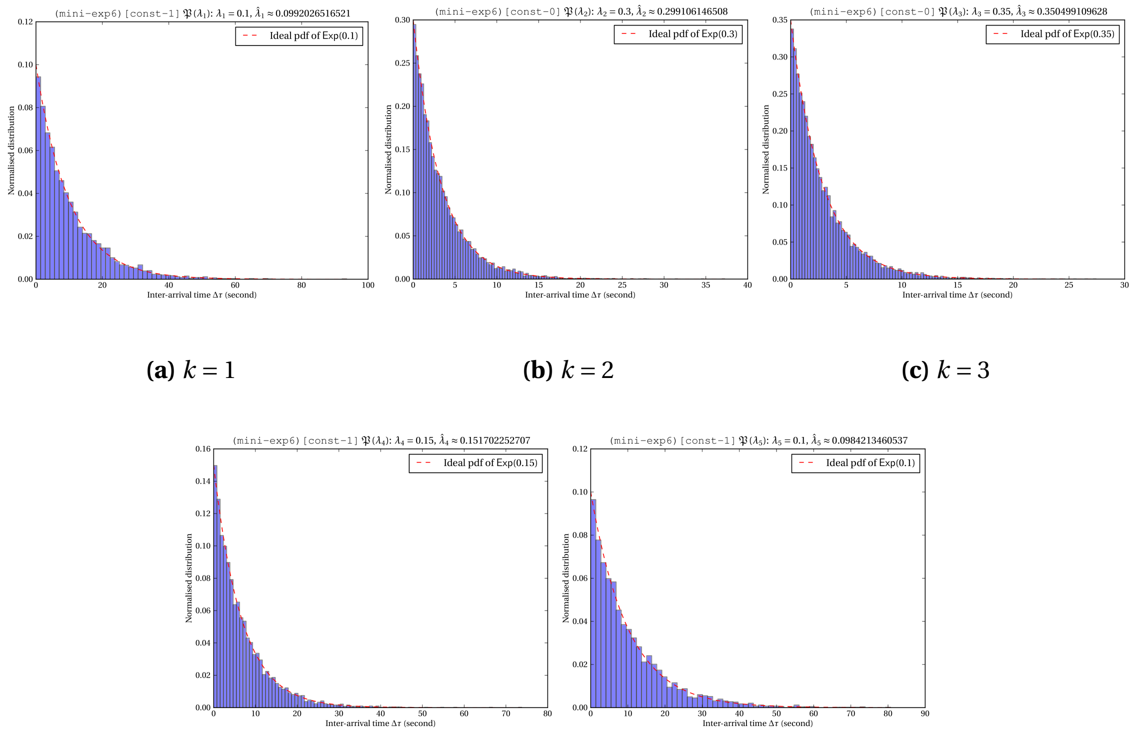
The **simulation package** was developed in Python [51] with intensive use of its scientific computing package SciPy [52] and plotting library matplotlib [53]. Backed by a MySQL relational database [54], the simulation directly implements the mathematical model of corollary 3.10. In fact, the package also supports the model of corollary 3.14 for use in a later empirical verification [section 6.2]. It allows flexible configurations of N (total number of input Poisson sequences), $\boldsymbol{\lambda}$ and $\boldsymbol{\xi}_v$ (mean arrival rates and characteristic functions of these constant sequences) as well as simulation duration $[0, t_{\max}]$.

A **simulation run** was launched for the concrete scenario of **example 3.7** [page 33] over a period of 12 hours in simulation time, i.e. $t_{\max} = 43{,}200$ seconds. Its outcomes are presented in figures 6.1 to 6.3 and table 6.1 and discussed in the subsequent paragraphs.

As can be seen in figure 6.1, the normalised **histogram of inter-arrival times** $\Delta\tau_{ki}$ of each input process $\mathfrak{P}(\lambda_k)$ closely matches the probability density function of exponential distribution $\mathsf{Exp}(\lambda_k)$, confirming the plausibility of these Poisson process generations. Indeed, inter-arrival time $\Delta\tau_i$ of a homogeneous Poisson process with mean arrival rate λ should be exponentially distributed with mean λ^{-1} [39, Interval Theorem, section 4.1, pp. 38–41]. In other words, $\Delta\tau_i \sim \mathsf{Exp}(\lambda)$ for $\mathfrak{P}(\lambda)$.

Figure 6.2 shows normalised histograms of inter-arrival times of observed processes and the ideal probability density functions of $\mathsf{Exp}(\lambda)$ and $\mathsf{Exp}\left(\tilde{\lambda}\right)$ with λ and $\tilde{\lambda}$ computed in example 3.7 [page 33]. The graphical matches demonstrate no discrepancies with the fact that these processes are indeed homogeneous Poisson with the respective rates [corollary 3.10].

All histograms employ Freedman-Diaconis rule [55] to determine the ‘best’ number of bins for



(a) $k = 1$

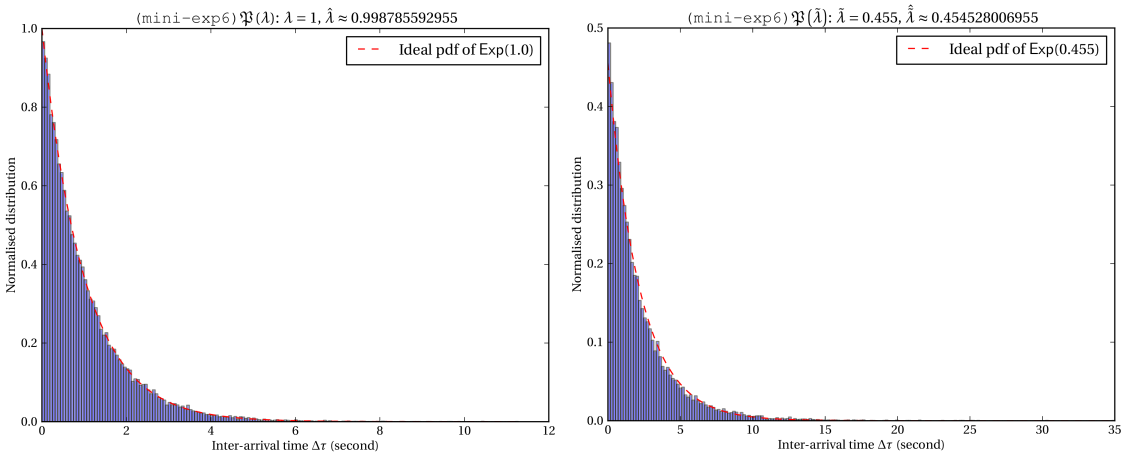
(b) $k = 2$

(c) $k = 3$

(d) $k = 4$

(e) $k = 5$

Figure 6.1: Normalised histograms and ideal probability density function of inter-arrival times $\Delta\tau_{ki} \sim \text{Exp}(\lambda_k)$ of controlled Poisson processes $\mathfrak{P}(\lambda_k)$ in the simulation of section 6.1



(a) Superposition $\mathfrak{P}(\lambda)$

(b) Different-consecutive-elements events $\mathfrak{P}(\tilde{\lambda})$

Figure 6.2: Normalised histograms and ideal probability density function of inter-arrival times $\Delta\tau_i \sim \text{Exp}(\lambda)$ and $\Delta\tilde{\tau}_i \sim \text{Exp}(\tilde{\lambda})$ of observed Poisson processes $\mathfrak{P}(\lambda)$ and $\mathfrak{P}(\tilde{\lambda})$ respectively in the simulation of section 6.1

inter-arrival time samples $\{\Delta\tau_i\}$:

$$\#\text{bins}\,\{\Delta\tau_i\} = \frac{\left(\max_i \Delta\tau_i - \min_i \Delta\tau_i\right)\sqrt[3]{n}}{2 \times \text{IQR}\,\{\Delta\tau_i\}} \tag{6.1}$$

where $n = |\{\Delta\tau_i\}|$ is the sample size and $\text{IQR}\,\{\Delta\tau_i\}$ denotes the sample inter-quartile range.

The **experimental outcomes** of both input (controlled) and output (observed) Poisson processes are also summarised in table 6.1 with the following details:

- *Rates* — Nominal values from either input configurations (for controlled processes $\{\mathfrak{P}(\lambda_k)\}$) or applications of corollary 3.10 in example 3.7 [page 33] (for observed processes — superposition $\mathfrak{P}(\lambda)$ and new $\mathfrak{P}(\tilde{\lambda})$ formed by different-consecutive-elements events in superposition sequence $\langle s_j \rangle \sim \mathfrak{P}(\lambda)$).
- *Sample sizes* — Total number of inter-arrival time samples $\Delta\tau_i \sim \mathsf{Exp}(\lambda)$ generated by the respective process $\mathfrak{P}(\lambda)$ throughout the simulation duration $[0, t_{\max}]$.
- *Rate estimates* — Maximum likelihood estimation $\hat{\lambda}$ for distribution $\mathsf{Exp}(\lambda)$ based on n inter-arrival time samples $\{\Delta\tau_i\}$ of the corresponding process $\mathfrak{P}(\lambda)$:

  $$\hat{\lambda} = \frac{n}{\sum_i \Delta\tau_i} \tag{6.2}$$

  where $n = |\{\Delta\tau_i\}|$ is the sample size.
- 95% *confidence intervals* — Based on the confidence interval of the underlying inter-arrival time $\Delta\tau_i \sim \mathsf{Exp}(\lambda)$ [56, section 7.6, pp. 267–268], the $100(1-\alpha)\%$ confidence interval ($\alpha \in [0,1]$) of arrival rate λ of $\mathfrak{P}(\lambda)$ is:

  $$\left[\frac{\text{qf}_{\mathsf{ChiSq}(2n)}\left(\frac{\alpha}{2}\right)}{2n}\hat{\lambda}, \frac{\text{qf}_{\mathsf{ChiSq}(2n)}\left(1-\frac{\alpha}{2}\right)}{2n}\hat{\lambda}\right] \tag{6.3}$$

  where $n = |\{\Delta\tau_i\}|$ is the sample size and $\text{qf}_{\mathsf{ChiSq}(m)}(p)$ is the quantile function evaluated at $p \in [0,1]$ of a χ^2 distribution with m degrees of freedom. In this case, equation (6.3) was applied with $\alpha = 0.05$ to obtain the 95% confidence intervals.

In order to inspect the **evolution of arrival rates** over time, statistical analyses similar to table 6.1 can be performed over a reduced look-back window of n inter-arrival time samples at every time point t. This is instead of considering the entire sample set $\{\Delta\tau_i\}$ corresponding to the whole simulation duration $[0, t_{\max}]$. The window-based approximation attempts to simulate a 'rate meter' which continually evaluates the 'instantaneous' arrival rate at time t. Such meter remains hypothetical and hence approximate as Poisson arrivals are inherently discrete events, albeit over a continuous timeline.

Table 6.1: Arrival rate estimates, sample sizes and 95% confidence intervals of rate estimates for homogeneous Poisson processes in the simulation of section 6.1

Process	*Type*	*Rate*	*Rate estimate*	*Sample size*	95% *confidence interval*
$\mathfrak{P}(\lambda_1)$	controlled	$\lambda_1 = 0.1$	$\hat{\lambda}_1 \approx 0.0992027$	$n_1 = 4281$	$[0.0962530, 0.1021962]$
$\mathfrak{P}(\lambda_2)$	controlled	$\lambda_2 = 0.3$	$\hat{\lambda}_2 \approx 0.2991061$	$n_2 = 12919$	$[0.2939704, 0.3042858]$
$\mathfrak{P}(\lambda_3)$	controlled	$\lambda_3 = 0.35$	$\hat{\lambda}_3 \approx 0.3504991$	$n_3 = 15140$	$[0.3449380, 0.3561041]$
$\mathfrak{P}(\lambda_4)$	controlled	$\lambda_4 = 0.15$	$\hat{\lambda}_4 \approx 0.1517023$	$n_4 = 6553$	$[0.1480512, 0.1553971]$
$\mathfrak{P}(\lambda_5)$	controlled	$\lambda_5 = 0.1$	$\hat{\lambda}_5 \approx 0.0984213$	$n_5 = 4251$	$[0.0954847, 0.1014019]$
$\mathfrak{P}(\lambda)$	observed	$\lambda = 1$	$\hat{\lambda} \approx 0.9987856$	$n = 43144$	$[0.9893830, 1.0082320]$
$\mathfrak{P}(\tilde{\lambda})$	observed	$\tilde{\lambda} = 0.455$	$\hat{\tilde{\lambda}} \approx 0.4545280$	$\tilde{n} = 19634$	$[0.4481922, 0.4609077]$

The outcomes of this rate evolution inspection are plotted in figure 6.3, with 95% confidence intervals indicated in light colour shades for observed processes. The estimates are unsurprisingly fluctuating around the theoretical value for smaller look-back window size n, exhibiting wider confidence intervals [equation (6.3)]. At the same time, larger n gives more stable results which are relatively closer to the theoretical values calculated in example 3.7 [page 33] by applying corollary 3.10. As all Poisson processes involved are homogeneous, there are no major rate shifts other than minor fluctuations around a stable mean.

The aforementioned simulation and empirical analyses were repeated for different values of N, $\boldsymbol{\lambda}$ and $\boldsymbol{\xi}_v$, all of which yielded similarly consistent outcomes with regard to corollary 3.10. Details of these other simulation runs are thus omitted for the sake of brevity.

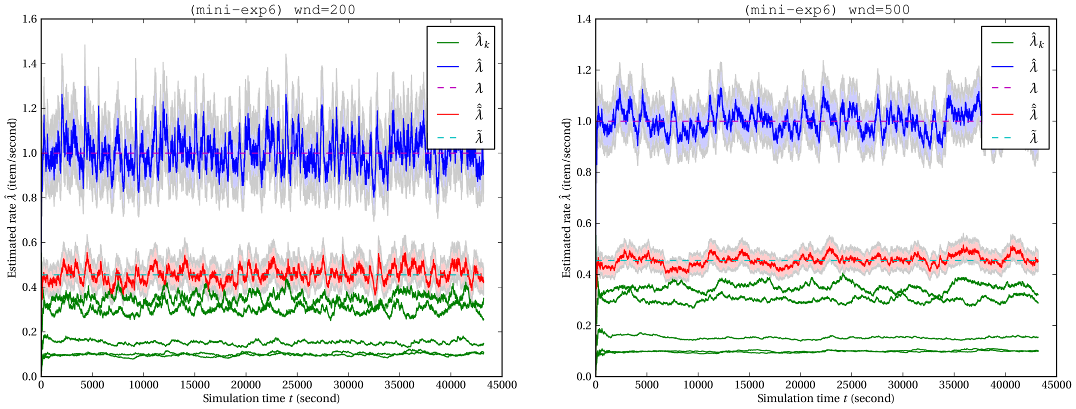
6.2 Empirical verification of corollary 3.14

Computer simulation was also employed to empirically verify **corollary 3.14** on the superposition of α-thinned alternating binary Poisson sequences, a major foundation for non-first-layer rate transfer analyses [section 4.5].

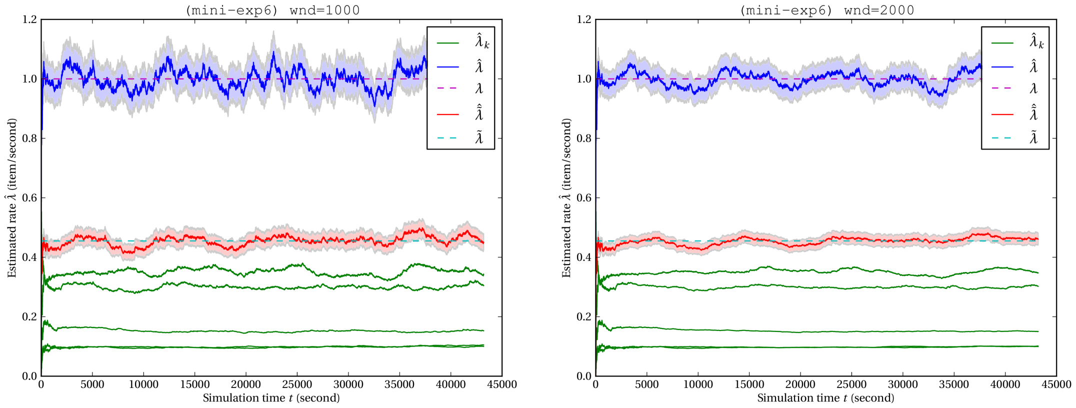
In fact, the same **simulation package** as that of section 6.1 was used as it was designed to be compatible with the mathematical models of both corollaries 3.10 and 3.14, allowing either constant or alternating binary sequences as inputs. All configuration options remain intact except that characteristic functions $\boldsymbol{\xi}_v$ are irrelevant and thinning probability α [definition 3.5] needs to be supplied in the latter case.

A **simulation run** was launched for the concrete scenario of **example 3.8** [page 36] over a period of 12 hours in simulation time. Its outcomes are presented in figures 6.4 to 6.6 and table 6.2. Only main conclusions from these results are summarised below as all observations and discussions are deemed analogous to those of section 6.1.

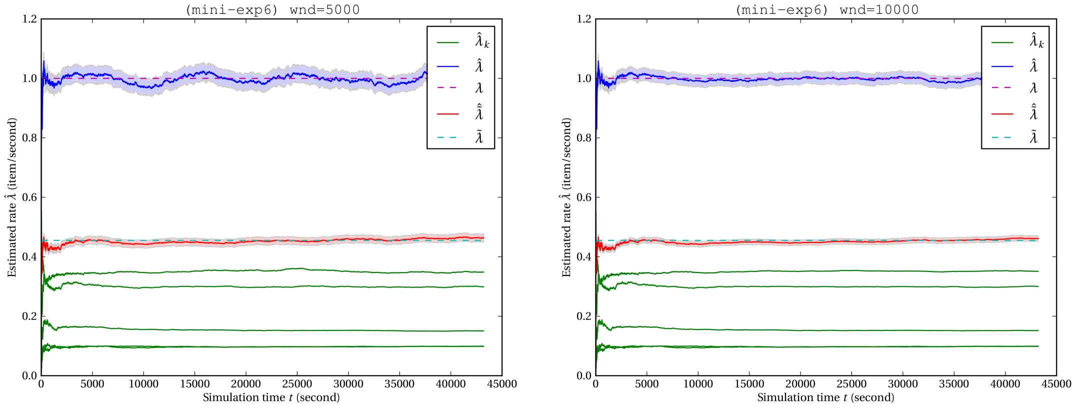
First of all, the plausibility of Poisson process generations were confirmed through the **histograms** of figure 6.4. Furthermore, figure 6.5 demonstrates no discrepancies with the fact



(a) $n = 200$ **(b)** $n = 500$

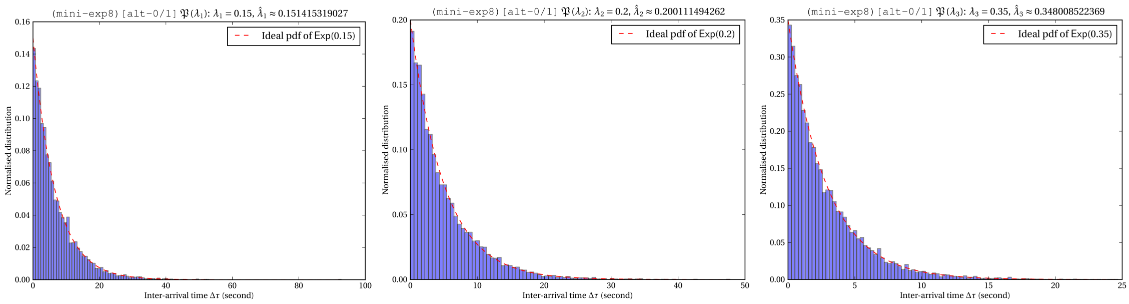


(c) $n = 10^3$ **(d)** $n = 2 \times 10^3$

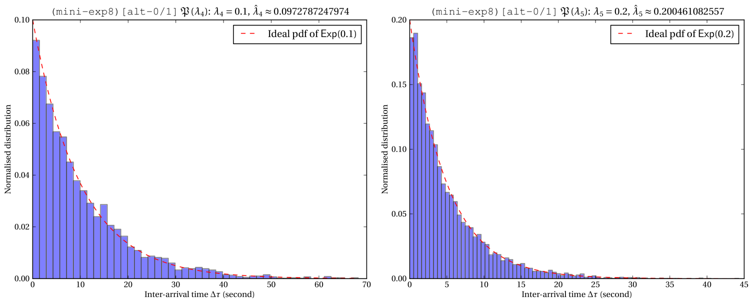


(e) $n = 5 \times 10^3$ **(f)** $n = 10^4$

Figure 6.3: Arrival rate estimates $\hat{\lambda}_k$, $\hat{\lambda}$ and $\hat{\hat{\lambda}}$ of Poisson processes $\mathfrak{P}(\lambda_k)$, $\mathfrak{P}(\lambda)$ and $\mathfrak{P}(\tilde{\lambda})$ in the simulation of section 6.1, using different look-back window sizes n. For $\hat{\lambda}$ and $\hat{\hat{\lambda}}$, light colour shades indicate 95% confidence intervals.



(a) $k = 1$ **(b)** $k = 2$ **(c)** $k = 3$



(d) $k = 4$ **(e)** $k = 5$

Figure 6.4: Normalised histograms and ideal probability density function of inter-arrival times $\Delta\tau_{ki} \sim \mathrm{Exp}(\lambda_k)$ of controlled Poisson processes $\mathfrak{P}(\lambda_k)$ in the simulation of section 6.2

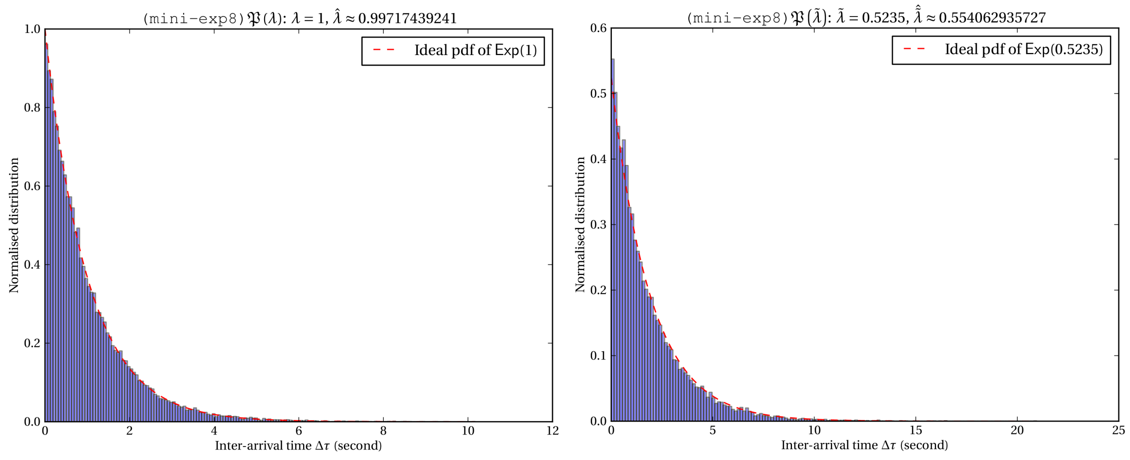
that observed processes are homogeneous Poisson with the rates affirmed by corollary 3.14. All histograms employ Freedman-Diaconis rule [55] to determine the ‘best’ number of bins [equation (6.1)]. Similar to table 6.1 whose description can be found on page 61, table 6.2 summarises the **experimental results**.

In addition, **rate evolution** was inspected just as in section 6.1 with results presented in figure 6.6. Same observations can be drawn — no major fluctuations due to homogeneity and closer results to the theoretical calculations of example 3.8 [page 36] for larger window size.

Lastly, the experiment was repeated for varying input configurations, all of which yielded consistent results with corollary 3.14. Details of these extra simulation runs can therefore be skipped for the sake of conciseness.

6.3 Prototypes of topology determination approaches

Apart from empirical verifications [sections 6.1 and 6.2], working prototypes of various topology determination approaches [chapter 5] were also developed in this study. Python [51] was selected as the environment of choice due to its portability and flexibility.

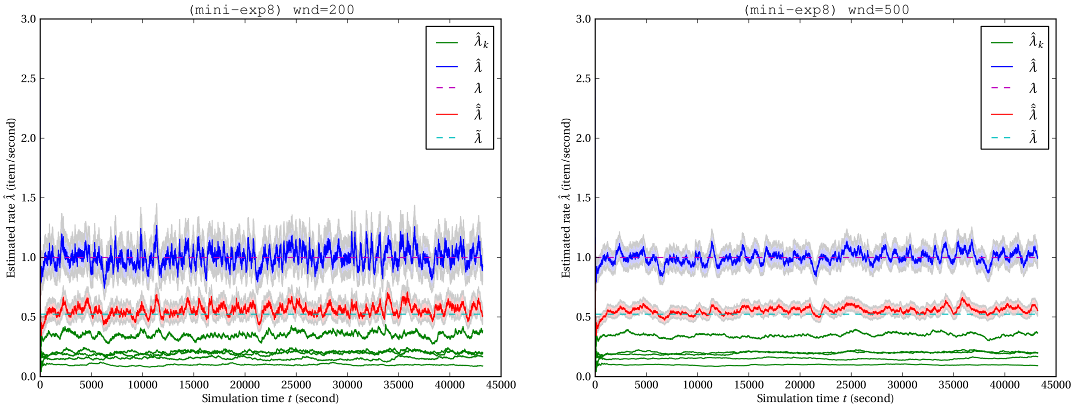


(a) Superposition $\mathfrak{P}(\lambda)$ **(b)** Different-consecutive-elements events $\mathfrak{P}(\tilde{\lambda})$

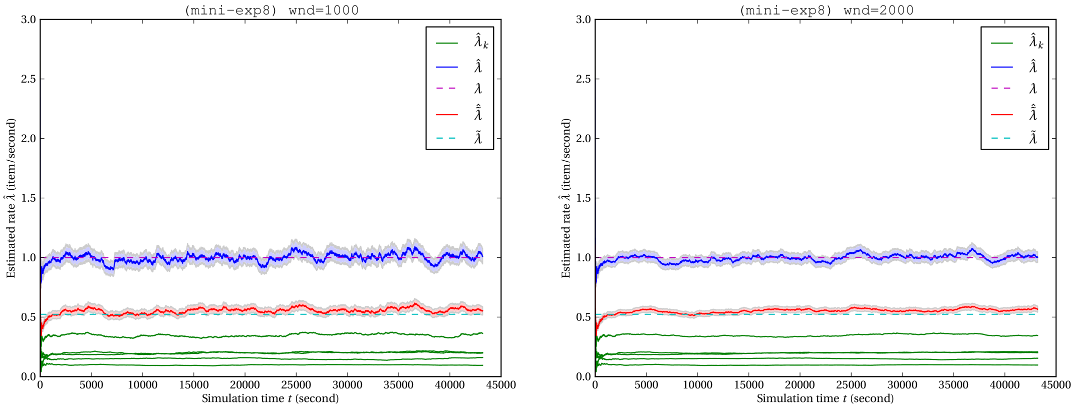
Figure 6.5: Normalised histograms and ideal probability density function of inter-arrival times $\Delta\tau_i \sim \mathrm{Exp}(\lambda)$ and $\Delta\tilde{\tau}_i \sim \mathrm{Exp}(\tilde{\lambda})$ of observed Poisson processes $\mathfrak{P}(\lambda)$ and $\mathfrak{P}(\tilde{\lambda})$ respectively in the simulation of section 6.2

Table 6.2: Arrival rate estimates, sample sizes and 95% confidence intervals of rate estimates for homogeneous Poisson processes in the simulation of section 6.2

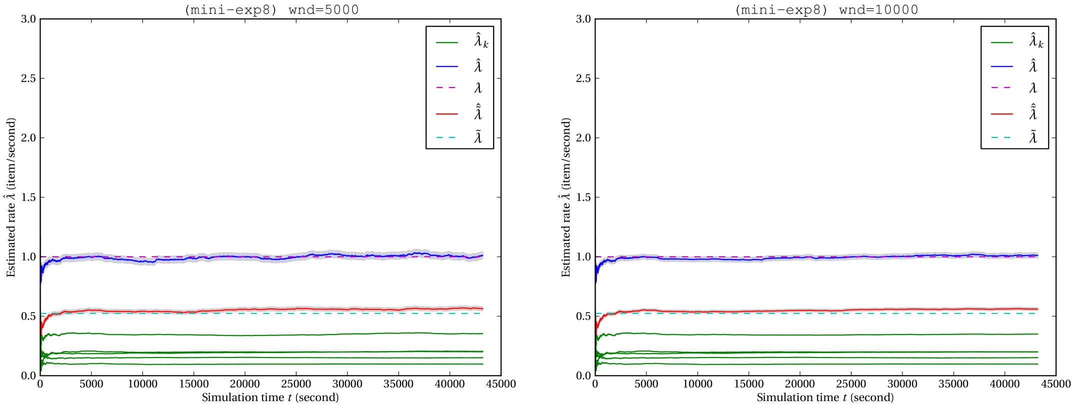
Process	*Type*	*Rate*	*Rate estimate*	*Sample size*	95% *confidence interval*
$\mathfrak{P}(\lambda_1)$	controlled	$\lambda_1 = 0.15$	$\hat{\lambda}_1 \approx 0.1514153$	$n_1 = 6541$	$[0.1477679, 0.1551066]$
$\mathfrak{P}(\lambda_2)$	controlled	$\lambda_2 = 0.2$	$\hat{\lambda}_2 \approx 0.2001115$	$n_2 = 8643$	$[0.1959147, 0.2043522]$
$\mathfrak{P}(\lambda_3)$	controlled	$\lambda_3 = 0.35$	$\hat{\lambda}_3 \approx 0.3480085$	$n_3 = 15033$	$[0.3424674, 0.3535935]$
$\mathfrak{P}(\lambda_4)$	controlled	$\lambda_4 = 0.1$	$\hat{\lambda}_4 \approx 0.0972787$	$n_4 = 4201$	$[0.0943591, 0.1002422]$
$\mathfrak{P}(\lambda_5)$	controlled	$\lambda_5 = 0.2$	$\hat{\lambda}_5 \approx 0.2004611$	$n_5 = 8659$	$[0.1962608, 0.2047052]$
$\mathfrak{P}(\lambda)$	observed	$\lambda = 1$	$\hat{\lambda} \approx 0.9971744$	$n = 43077$	$[0.9877797, 1.0066129]$
$\mathfrak{P}(\tilde{\lambda})$	observed	$\tilde{\lambda} = 0.5235$	$\hat{\tilde{\lambda}} \approx 0.5540629$	$\tilde{n} = 23935$	$[0.5470656, 0.5611041]$



(a) $n = 200$ **(b)** $n = 500$



(c) $n = 10^3$ **(d)** $n = 2 \times 10^3$



(e) $n = 5 \times 10^3$ **(f)** $n = 10^4$

Figure 6.6: Arrival rate estimates $\hat{\lambda}_k$, $\hat{\lambda}$ and $\hat{\hat{\lambda}}$ of Poisson processes $\mathfrak{P}(\lambda_k)$, $\mathfrak{P}(\lambda)$ and $\mathfrak{P}(\tilde{\lambda})$ in the simulation of section 6.2, using different look-back window sizes n. For $\hat{\lambda}$ and $\hat{\hat{\lambda}}$, light colour shades indicate 95% confidence intervals.

For **priority-based brute-force** approaches [section 5.3], a single prototype was implemented for both algorithms 5.1 and 5.2. The user can configure the number of sources n_0 [definition 2.1], mean arrival rate of each source λ_0 [definition 2.2] and the ingest rate constraint at each aggregation node θ [definition 2.7]. All of these constitute inputs to the topology determination problem as discussed in chapter 5. An additional configuration for the maximum number of aggregation nodes N_{max} is also required, as indicated in algorithm 5.2. For convenience purposes, N_{max} is also applicable to algorithm 5.1 as a simple stop condition. Algorithm selection is governed by a 'priority' parameter which can be either 'number of aggregation nodes' (for algorithm 5.1) or 'number of layers' (for algorithm 5.2).

For the **L-priority MINLP-based** approach [section 5.4], two separate prototypes were developed, both based on algorithm 5.3. One employs MIDACO [50] as the black-box MINLP optimiser while the other incorporates SCIP [46] for solving MINLP. The choices of MIDACO and SCIP were purely technical and convenient. Both have Python APIs (natively offered for MIDACO and through wrapper module python-zipopt [57] for SCIP) available for non-commercial use. On the whole, adaptation was required according to each solver's specification (e.g. how constraints should be passed); however, this does not change the nature of our MINLP problem. The outer loop of algorithm 5.3 with incremental-L was not included and should rather be constructed at the invocation script level for simplicity.

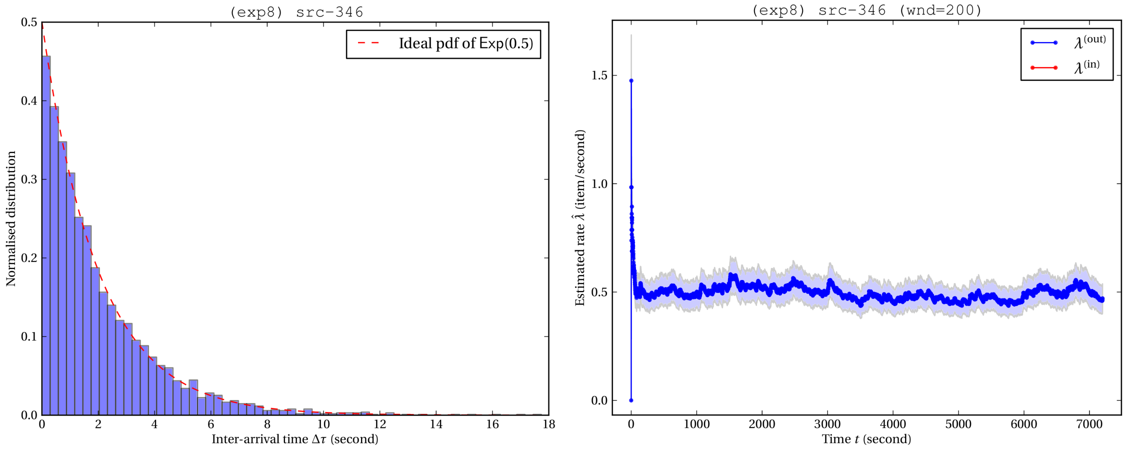
Owing to the non-commercial editions of MIDACO and SCIP, our prototypes unfortunately suffer from significant drawbacks in terms of problem scale. In particular, non-commercial MIDACO only allows up to four optimisation variables (directly limiting our number of layers L) while its counterpart SCIP permits only bilinear constraints (indirecting limiting L as well, cf. equation (5.1)). Consequently, no prototypes have been provided for the **N-cap MINLP-based** approach [section 5.6] as algorithm 5.5 inherently requires a much larger MINLP problem scale than that of algorithm 5.3.

For **conservative sequential assignments** [section 5.5], a prototype was also developed based on algorithm 5.4. This prototype has exactly the same interface as that of its brute-force counterparts described above. It is however much more efficient that brute-force searches and does not suffer from non-commercial MINLP solvers' constraints. As a result, this prototype will be used to determine the topology for subsequent experiments.

6.4 Simulation of multi-layer parallelisation

A complete simulation package was developed to study the overall behaviour of source and aggregation nodes at various layers of the parallelisation [definition 2.1].

Just as the smaller-scale simulations of sections 6.1 and 6.2, this **package** was implemented in Python [51] using scientific computing package SciPy [52], plotting library matplotlib [53] and a MySQL relational database [54] to simulate all aspects of the formalisation [section 2.2]. Configurations for a simulation run include:



(a) Inter-arrival times

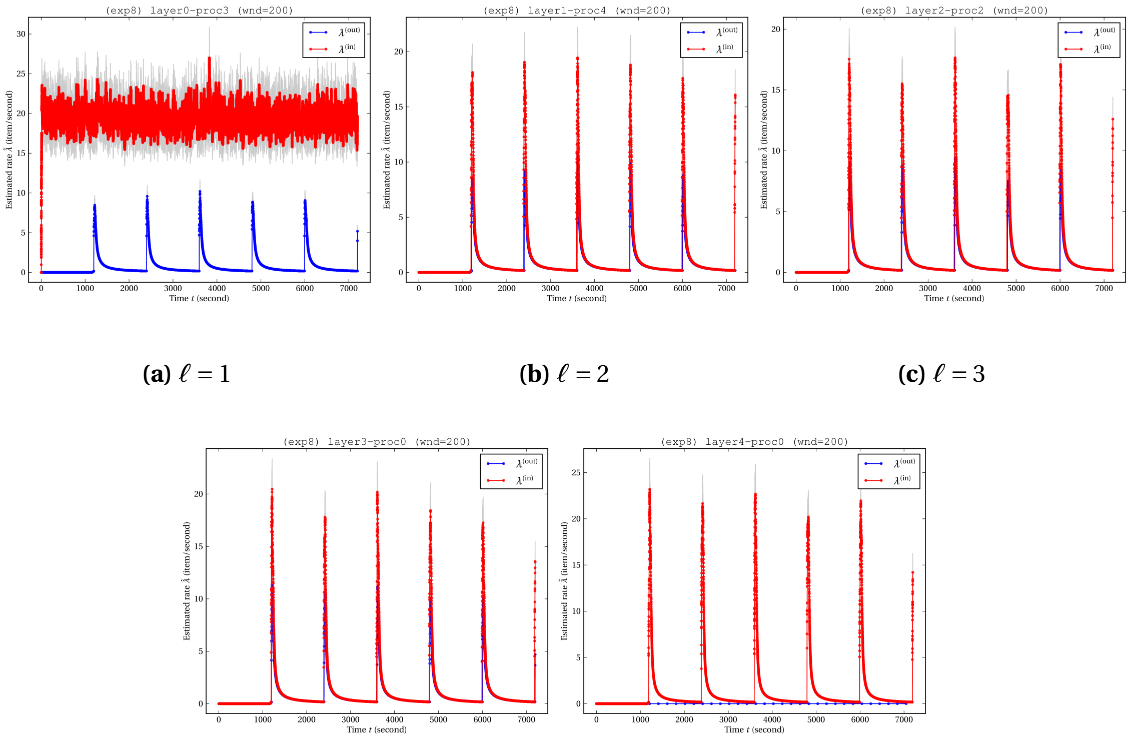
(b) Rate estimates (window size 200)

Figure 6.7: Behaviour at a randomly selected layer-0 source node in the simulation of section 6.4

- $[0, t_{\max}]$: Simulation duration.
- n_0: Number of sources [definition 2.2].
- $\mathbf{n} = [n_\ell] \in (\mathbb{N}^*)^L$: Numbers of aggregation nodes [definition 2.3] at layers $\ell \in \{1..L\}$.
- λ_0: Poisson mean arrival rate at each source [definition 2.2].
- Δt: Ideal key-change period at each source [definition 2.2].
- σ: Standard deviation for key-change offset Ω_k at each source [definition 2.2].
- agg(.): Admissible aggregate function [definition 1.4] – min(.), max(.), sum(.) or count(.).
- Range $\{v_{\min}..v_{\max}\}$ for random integer target values v in tuples (T, v) output by sources.
- Propagation delay induced between any two layers.

It is noteworthy that the last three configurations, while included for completeness, are irrelevant for our rate-focused analyses, cf. chapter 4. We then experimented with a concrete **scenario** over a period of 2 hours in simulation time ($t_{\max} = 7{,}200$ seconds): $n_0 = 500$ sources, $\lambda_0 = 0.5$ items/sec, $\Delta t = 1{,}200$ seconds, $\sigma = 24$ seconds, sum(.) as aggregation function, $v_{\min} = 0$, $v_{\max} = 99$ and 2-second propagation delay. Assuming an ingest rate constraint $\theta = 20$ items/sec [definition 2.7] at each aggregation node, executing the topology determination prototype [section 6.3] based on conservative sequential assignments [section 5.5] yielded a 5-layer topology with sizes $\mathbf{n} = [n_\ell] = \begin{bmatrix} 13 & 7 & 4 & 2 & 1 \end{bmatrix}^\mathsf{T}$. The simulation was conducted with the above scenario and this $\mathbf{n}$.

Experimental **outcomes** of the run are presented in figures 6.7 and 6.8 for source and aggregation layers respectively. Indeed, Poisson process generations at the prescribed rate $\lambda_0 = 0.5$ by *sources* were confirmed through inter-arrival time histograms and evolution of rate estimates, shown in figure 6.7 for a randomly selected source. Just as in sections 6.1 and 6.2, Freedman-Diaconis rule [55] was used to identify the number of histogram bins and rate evolution was based on maximum likelihood estimation over a look-back window.



(a) $\ell = 1$ **(b)** $\ell = 2$ **(c)** $\ell = 3$

(d) $\ell = 4$ **(e)** $\ell = 5$

Figure 6.8: Rate estimates at randomly selected layer-ℓ aggregation nodes in the simulation of section 6.4 (look-back window size 200)

For *aggregation layers*, evolution plots of rate estimates were also obtained, as shown in figure 6.8 for one randomly selected node per layer. Besides the constant incoming rate at layer 1, all other incoming and outgoing rates [definition 2.6] exhibited obvious 'spikes' around ideal key-change moments $\left\{t_i^{(\text{ideal})}\right\}$, i.e. every $\Delta t = 1{,}200$ seconds. This is in line with the analyses of sections 4.4 and 4.5. However, these rates are dynamic and fast-changing (piecewise constant with small constant-rate time windows). Therefore, measured amplitudes in the evolution could be unreliable due to insufficient samples for maximum likelihood estimation (cf. section 6.1). In fact, they are sensitive to the look-back window size (cf. figures 6.3 and 6.6).

The simulation was repeated for different configuration values and gave similar outcomes. Details of these other runs are therefore not presented.

6.5 Amazon Kinesis-based prototype

In order to demonstrate the practicality and applicability of proposed approaches, prototyping and experiments were conducted in the real Amazon Kinesis [section 1.4] environment. The **prototype suite** consists of four Java [26] applications — `CreateStreamApp`, `SourceApp`, `ProcessorApp` and `SinkApp`. For experimental purposes, all applications log to a MySQL [54] relational database for post-analysis. As the names imply, administrative `CreateStreamApp` is responsible for Kinesis stream creation, while the rest correspond directly to sources [definition 2.2] and aggregation nodes [definition 2.3] of our multi-layer parallelisation [definition 2.1]. Runtime **interactions** of these prototype applications with the environment are depicted in the diagram of figure 6.10.

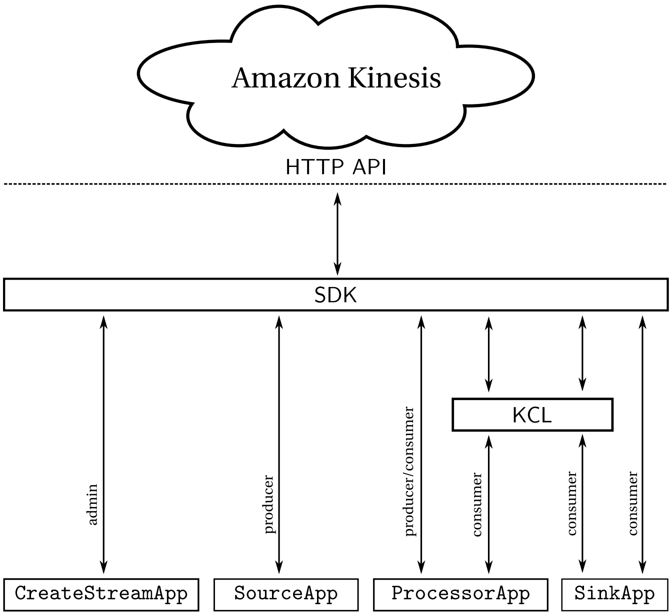


Figure 6.9: Interactions of prototype applications with Amazon Kinesis

Indeed, the applications follow the typical Kinesis usage scenario of figure 1.3 [section 1.4]. `SourceApp` is a single-role producer, `ProcessorApp` is a dual-role producer-consumer and `SinkApp` is a single-role consumer. As explained in section 1.4, producers interact with Kinesis via the SDK while consumers can employ KCL for push-model data consumption and the SDK for other operations. With many shared functionalities amongst the applications, object-oriented inheritance was employed to ensure maintainability and extensibility.

These applications realise the way Kinesis can be employed for time-based stream aggregation, as described in section 1.6. Consequently, their **deployment** in the Kinesis environment resembles the structure sketched in figure 6.10. In essence, every aggregation layer comprises a Kinesis stream with as many shards as the number of aggregation nodes, each being a single-worker consumer integrated in dual-role `ProcessorApp` or single-role `SinkApp`. Furthermore, uniform data spread to the next layer is automatically regulated by Kinesis shard partitioning scheme. Once the system has stabilised, KCL mechanism ensures that there is a one-to-one correspondence between shards and consumer.

In terms of **parameters**, apart from technical ones like AWS region ID, service endpoint and HTTP proxy, each application needs to be configured with an experiment duration $[0, t_{\max}]$, a node ID and relevant stream names. Specifically, outgoing and incoming stream names apply

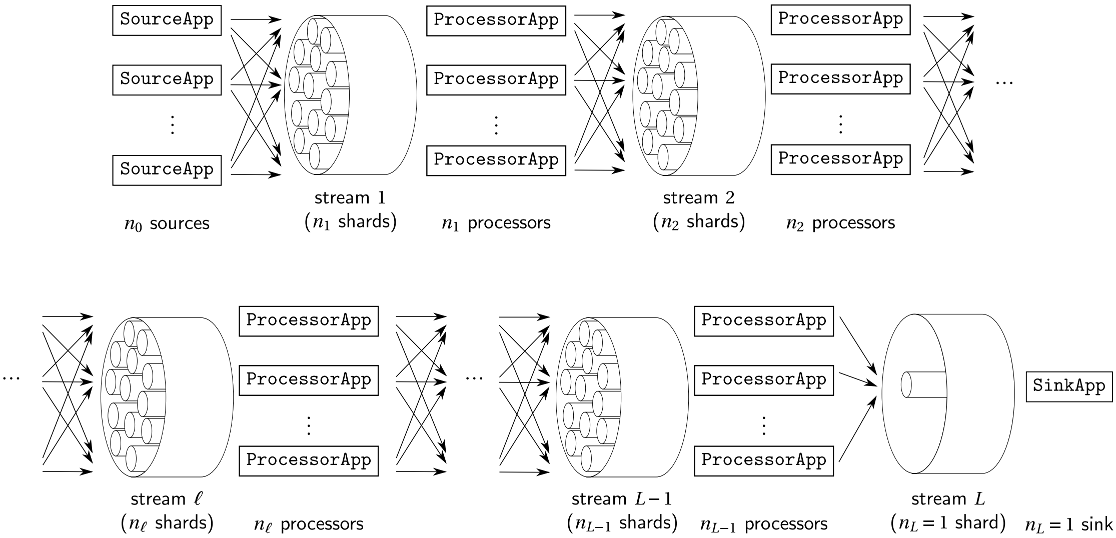


Figure 6.10: Deployment of prototype applications and Amazon Kinesis streams

respectively to `SourceApp` and `SinkApp` while both have to be provided for `ProcessorApp` (cf. figure 6.10). Moreover, emission rate λ_0 and ideal key-change period Δt are required for proper data generation of the `SourceApp`.

We also developed a Python [51] script to instantiate all applications and orchestrate their instances according to a given topology by 'chaining' them through correctly indicated incoming and/or outgoing stream names. In particular, this script takes in similar parameters as that of the simulation in section 6.4:

- $[0, t_{\max}]$: Experiment duration.
- n_0: Number of sources [definition 2.2].
- $\mathbf{n} = [n_\ell] \in (\mathbb{N}^*)^L$: Numbers of aggregation nodes [definition 2.3] at layers $\ell \in \{1..L\}$.
- λ_0: Data emission rate at each source [definition 2.2].
- Δt: Key-change period at each source [definition 2.2].
- agg(.): Admissible aggregate function [definition 1.4] – min(.), max(.), sum(.) or count(.).
- Technical parameters: AWS region ID, Kinesis endpoint, HTTP proxy, etc.

Unlike the simulation of section 6.4, each running `SourceApp` instance emits data tuples and switch time period IDs at pseudo-regular intervals. Natural randomness is added by real system and network conditions rather than artificially controlled Poisson processes.

For experimental purpose, our `SourceApp` emits data payload at the maximum size of 50 KB each [29] (padding is done when necessary). Considering the service limits of Kinesis, we approximately worked out the **ingest rate constraint** [definition 2.7] $\theta = 20$ items/sec based on the bottleneck of 1 MB written per second and the maximum payload size of 50 KB [29]. These tight configurations with respect to Kinesis limits allow us to experiment a 'saturation threshold' scenario. In practice, extra tolerance should be added to θ for additional safety while data payloads may not necessary be all of the maximum size. In fact, the maximum-size assumption is another way to add safety buffer to θ. Moreover, in production systems, empirical assessments can be conducted to determine θ instead of relying completely on specifications published by the service provider.

We then experimented with a concrete **scenario** during ≈ 1.1 hours ($t_{\max} = 4,000$ seconds): $n_0 = 500$ sources, $\lambda_0 = 0.5$ items/sec, $\Delta t = 1,200$ seconds and sum(.) as aggregation function. Using $\theta = 20$ items/sec derived above, running the topology determination prototype [section 6.3] based on conservative sequential assignments [section 5.5] yielded a 5-layer **topology** with sizes $\mathbf{n} = [n_\ell] = \begin{bmatrix} 13 & 7 & 4 & 2 & 1 \end{bmatrix}^\mathsf{T}$. In other words, we are considering a similar scenario to that of section 6.4 but now in the real Kinesis environment where natural randomness exists and true service limits apply.

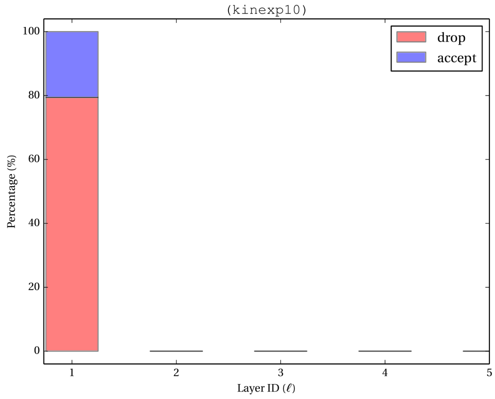
To facilitate comparison, five **experiments** were launched with the aforementioned topology as well as modified versions of it. These variations attempt to reduce the number of layers and terminate early with a single sink. In particular, the following topologies were experimented in the real Amazon Kinesis environment:

- (a) $\mathbf{n} = \begin{bmatrix} 1 \end{bmatrix}$
- (b) $\mathbf{n} = \begin{bmatrix} 13 & 1 \end{bmatrix}^\mathsf{T}$
- (c) $\mathbf{n} = \begin{bmatrix} 13 & 7 & 1 \end{bmatrix}^\mathsf{T}$
- (d) $\mathbf{n} = \begin{bmatrix} 13 & 7 & 4 & 1 \end{bmatrix}^\mathsf{T}$
- (e) $\mathbf{n} = \begin{bmatrix} 13 & 7 & 4 & 2 & 1 \end{bmatrix}^\mathsf{T}$

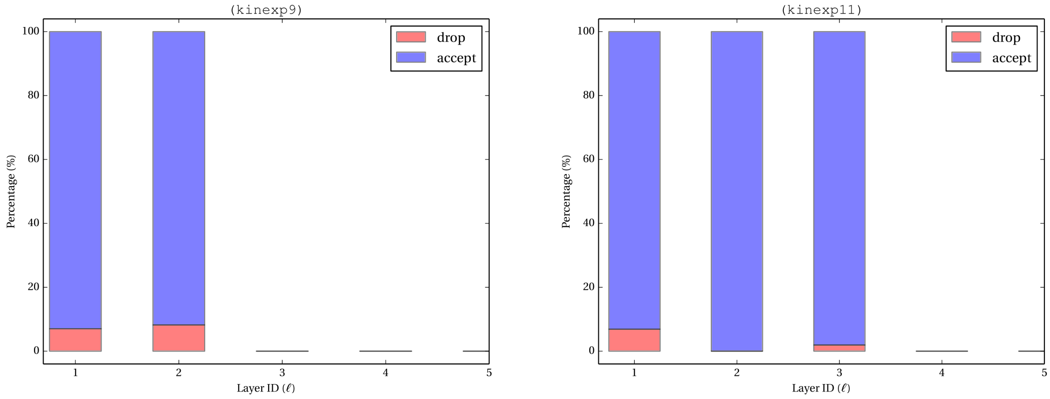
While all previous experiments had been conducted on an in-house Linux computing server, these five were launched on cloud-based Amazon EC2 [17] due to more demanding resource requirements by the experiments' scale. The **outcomes** of these experiments are presented in figure 6.11. Basically, we would like to observe the amount *dropped data* upon their arrivals at various aggregation layers (due to *ingest rate saturation* at aggregation nodes).

We noted that the number of data tuples and their significance levels vary greatly from one layer to another. For instance, layer 1 receives a considerably large amount of data emitted by a huge collection of sources (in our case, 500 sources versus tens of aggregation nodes). On the contrary, fewer data tuples arrive at subsequent layers $\ell > 1$ with substantially higher importance, for these are $(\ell - 1)^{\text{st}}$-level aggregates of several original tuples, cf. algorithm 2.1. In other words, $\forall \ell_1, \ell_2 \in \{1..L\}$ such that $\ell_1 < \ell_2$, a drop at layer ℓ_2 is more 'costly' for the final output than its counterpart at layer ℓ_1. Therefore, *drop percentages* (instead of absolute numbers) at individual aggregation node were considered for reasonable cross comparison.

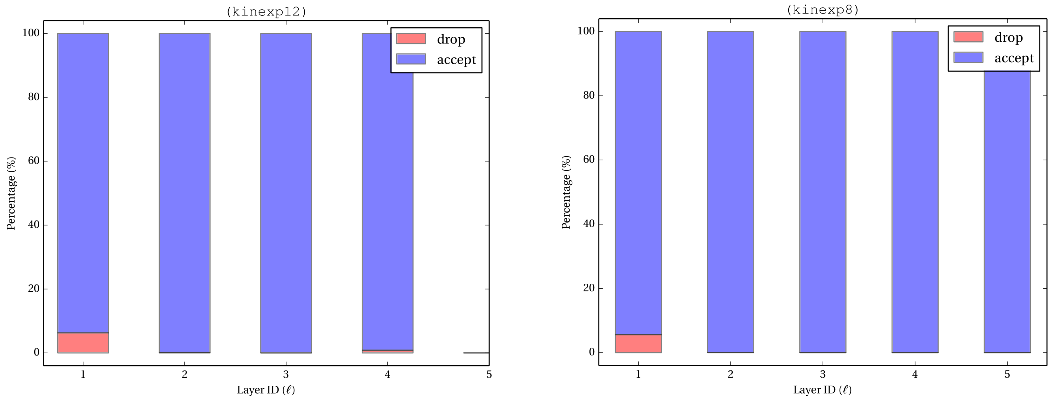
As can be seen from figure 6.11, the single sink layer in *topology (a)* suffered from a significantly high drop percentage of approximately 80%. This was unsurprisingly consistent because, according to theorem 4.2, the total incoming rate is $\lambda_1^{(\text{in})} = \dfrac{n_0 \lambda_0}{n_1} = \dfrac{500 \times 0.5}{1} = 250 \gg 20 = \theta$.



(a) $\mathbf{n} = \begin{bmatrix}1\end{bmatrix}$



(b) $\mathbf{n} = \begin{bmatrix}13 & 1\end{bmatrix}^\mathsf{T}$ **(c)** $\mathbf{n} = \begin{bmatrix}13 & 7 & 1\end{bmatrix}^\mathsf{T}$



(d) $\mathbf{n} = \begin{bmatrix}13 & 7 & 4 & 1\end{bmatrix}^\mathsf{T}$ **(e)** $\mathbf{n} = \begin{bmatrix}13 & 7 & 4 & 2 & 1\end{bmatrix}^\mathsf{T}$

Figure 6.11: Percentages of accepted/dropped data in Kinesis-based experiments of section 6.5 – Each experiment is denoted by $\mathbf{n} = [n_\ell]$, numbers of nodes at various aggregation layers

With 13 processors, *topology (b)* exhibited a much lower drop percentage at layer 1. This was reduced from ≈ 80% to ≈ 5% but not absolute zero as ideally expected because we operated in a saturation threshold scenario based on published service limits. At sink layer 2, there were still some drops measured at around 6% of the total number of data arrivals.

As more aggregation layers were added in *topologies (c) and (d)*, there were a noticeable decrease (to almost zero) of the drop percentages at intermediate layers and also smaller drop percentages at the sink layer. The latter was then reduced to almost zero when the complete 5-layer *topology (e)* suggested by the topology determination prototype [section 6.3] was adopted. The reader is reminded that, drops at subsequent layers were considered more 'costly' in than those in preceding layers, as discussed above.

All in all, using the topology determined through a systematic approach turned out to be more *beneficial* in terms of the overall accuracy and completeness for our time-based stream aggregation. Moreover, such approach *minimises* the *total number of aggregation nodes* (read '*number of shards*' in the context of Amazon Kinesis, which forms the fundamental basis for billing [28], cf. section 1.4) as well as the *number of layers* (read '*number of streams*' in the Kinesis context, directly related to system complexity and output latency).

7 Concluding remarks

In sum, we successfully addressed the given problem statement [section 2.2] with solid analyses [chapter 4] and plausible solutions [chapter 5], supported by theoretical foundations [chapter 3] as well as empirical verifications [chapter 6]. In this concluding chapter, major contributions of the work are first highlighted [section 7.1] before some possible future work is suggested [section 7.2].

7.1 Contributions

In this thesis, we have studied the rate transfer properties at all layers of the multi-layer parallelisation of time-based stream aggregation, both analytically [chapters 3 and 4] and empirically [chapter 6]. To the best of the author's knowledge, no such studies had been conducted previously in this context.

Based on the above analyses, various systematic approaches to the determination of a topology for such multi-layer parallelisation have been proposed [chapter 5] in the context of ingest rate constraints present at aggregation nodes. These proposed approaches, given as both theoretical algorithmic methods [chapter 5] and practical prototype implementations [section 6.3], aim to optimise operational cost, output latency and system complexity. These are achieved by concurrently minimising the total number of aggregation nodes and the number of aggregation layers, while ensuring that no ingest rate saturation due to constraint violation occurs any aggregation nodes throughout the topology.

Relevant simulations and experiments have also been conducted [chapter 6] to verify system behaviours and the plausibility of the proposed topology determination methods. While the whole work has been motivated [section 1.6] by and experimented [section 6.5] on the Amazon Kinesis environment [section 1.4], all analyses and proposed approaches remain applicable in the general context [section 2.2] without being restricted to any specific platforms and adaptable to situations beyond the stated project scope [section 2.1].

7.2 Possible future work

There certainly remains room for improvement and extension in this study. Some possible candidates for future work include:

- Adaptation to dynamic/heterogeneous data emission rates at sources.
- Adaptation to dynamic/heterogeneous ingest rate constraints at aggregation nodes.
- Application to parallelisation schemes of local stream processing frameworks such as the computing-focused Storm [11].
- Reliable method to assess ingest rate constraints without complete reliance on published specifications from cloud service provider.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[2] Apache Software Foundation, "Welcome to Apache™Hadoop®!" 2014, [Accessed: March–July 2014]. [Online]. Available: http://hadoop.apache.org

[3] ——, "HDFS Architecture Guide," 2008, [Accessed: March–July 2014]. [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '02. New York, NY, USA: ACM, 2002, pp. 1–16. [Online]. Available: http://doi.acm.org/10.1145/543613.543615

[5] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *SIGMOD Rec.*, vol. 34, no. 4, pp. 42–47, Dec. 2005. [Online]. Available: http://doi.acm.org/10.1145/1107499.1107504

[6] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, "STREAM: The Stanford data stream management system," Stanford InfoLab, Technical Report 2004-20, 2004. [Online]. Available: http://ilpubs.stanford.edu:8090/641/

[7] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A new model and architecture for data stream management," *The VLDB Journal*, vol. 12, no. 2, pp. 120–139, Aug. 2003. [Online]. Available: http://dx.doi.org/10.1007/s00778-003-0095-z

[8] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the Borealis stream processing engine," in *In CIDR*, 2005, pp. 277–289. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.118.7039

[9] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "MillWheel: Fault-tolerant stream processing at internet

scale," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1033–1044, Aug. 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2536222.2536229

[10] R. Ananthanarayanan, V. Basker, S. Das, A. Gupta, H. Jiang, T. Qiu, A. Reznichenko, D. Ryabkov, M. Singh, and S. Venkataraman, "Photon: Fault-tolerant and scalable joining of continuous data streams," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 577–588. [Online]. Available: http://doi.acm.org/10.1145/2463676.2465272

[11] Apache Software Foundation, "Storm, distributed and fault-tolerant real-time computation," [Accessed: March–July 2014]. [Online]. Available: https://storm.incubator.apache.org

[12] Google Inc., "Google Cloud Platform," [Accessed: March–July 2014]. [Online]. Available: https://cloud.google.com

[13] Amazon Web Services Inc., "Amazon Web Services (AWS) – cloud computing services," 2014, [Accessed: March–July 2014]. [Online]. Available: http://aws.amazon.com

[14] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, Jun. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.future.2008.12.001

[15] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[16] Google Inc., "Google Compute Engine – cloud computing & infrastructure as a service – Google Cloud Platform," [Accessed: March–July 2014]. [Online]. Available: https://cloud.google.com/products/compute-engine

[17] Amazon Web Services Inc., "AWS | Amazon Elastic Compute Cloud (EC2) – scalable cloud hosting," 2014, [Accessed: March–July 2014]. [Online]. Available: http://aws.amazon.com/ec2

[18] ——, "AWS | Amazon Kinesis," 2014, [Accessed: March–July 2014]. [Online]. Available: http://aws.amazon.com/kinesis

[19] ——, "AWS documentation | Amazon Kinesis API reference," 2013, [Accessed: March–July 2014]. [Online]. Available: http://docs.aws.amazon.com/kinesis/latest/dev

[20] ——, "AWS documentation | Amazon Kinesis developer guide," 2013, [Accessed: March–July 2014]. [Online]. Available: http://docs.aws.amazon.com/kinesis/latest/APIReference

[21] D. Stenberg, "`curl` and `libcurl`," 2014, [Accessed: March–July 2014]. [Online]. Available: http://curl.haxx.se

[22] Apache Software Foundation, “Apache HttpComponents,” 2001–2011, [Accessed: March–July 2014]. [Online]. Available: http://hc.apache.org

[23] Amazon Web Services Inc., “AWS Kinesis management console,” 2008–2014, [Accessed: March–July 2014]. [Online]. Available: https://console.aws.amazon.com/kinesis

[24] ——, “AWS SDK for Java,” 2014, [Accessed: March–July 2014]. [Online]. Available: http://aws.amazon.com/sdkforjava

[25] ——, “AWS SDK for PHP,” 2014, [Accessed: March–July 2014]. [Online]. Available: http://aws.amazon.com/sdkforphp

[26] Oracle, “Oracle Technology Network for Java developers,” [Accessed: March–July 2014]. [Online]. Available: http://www.oracle.com/technetwork/java

[27] The PHP Group, “PHP: Hypertext preprocessor,” 2001–2014, [Accessed: March–July 2014]. [Online]. Available: http://php.net

[28] Amazon Web Services Inc., “AWS documentation | Amazon Kinesis | pricing,” 2013, [Accessed: March–July 2014]. [Online]. Available: http://aws.amazon.com/kinesis/pricing

[29] ——, “Amazon Kinesis sizes and limits,” 2013, [Accessed: March–July 2014]. [Online]. Available: http://docs.aws.amazon.com/kinesis/latest/dev/service-sizes-and-limits.html

[30] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994. [Online]. Available: http://books.google.com/books?id=6oHuKQe3TjQC

[31] GitHub Inc., “Amazon Kinesis Client – GitHub,” 2014, [Accessed: March–July 2014]. [Online]. Available: https://github.com/awslabs/amazon-kinesis-client

[32] Amazon Web Services Inc., “AWS | Amazon DynamoDB – NoSQL cloud database service,” 2014, [Accessed: March–July 2014]. [Online]. Available: http://aws.amazon.com/dynamodb

[33] GitHub Inc., “Amazon Kinesis Connector Library – GitHub,” 2014, [Accessed: March–July 2014]. [Online]. Available: https://github.com/awslabs/amazon-kinesis-connectors

[34] ——, “Amazon Kinesis Storm Spout – GitHub,” 2014, [Accessed: March–July 2014]. [Online]. Available: https://github.com/awslabs/amazon-kinesis-connectors

[35] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google File System,” in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP ’03. New York, NY, USA: ACM, 2003, pp. 29–43. [Online]. Available: http://doi.acm.org/10.1145/945445.945450

[36] L. Rizzo, “netmap: A novel framework for fast packet I/O,” in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC’12. Berkeley, CA, USA: USENIX Association, 2012, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342821.2342830

[37] O. Papapetrou, M. Garofalakis, and A. Deligiannakis, “Sketch-based querying of distributed sliding-window data streams,” *Proc. VLDB Endow.*, vol. 5, no. 10, pp. 992–1003, Jun. 2012. [Online]. Available: http://dx.doi.org/10.14778/2336664.2336672

[38] Q. Huang and P. P. Lee, “LD-Sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams.” INFOCOM, 2014.

[39] J. F. C. Kingman, *Poisson Processes*, ser. Oxford studies in probability. Clarendon Press, 1992. [Online]. Available: http://books.google.com/books?id=VEiM-OtwDHkC

[40] T. Kohno, A. Broido, and K. Claffy, “Remote physical device fingerprinting,” *Dependable and Secure Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 93–108, April 2005. [Online]. Available: http://dx.doi.org/10.1109/TDSC.2005.26

[41] A. Papoulis and S. Pillai, *Probability, random variables, and stochastic processes*, ser. McGraw-Hill electrical and electronic engineering series. McGraw-Hill, 2002. [Online]. Available: http://books.google.com/books?id=YYwQAQAAIAAJ

[42] M. R. Bussieck and A. Pruessner, “Mixed-integer nonlinear programming,” *SIAG/OPT Newsletter: Views & News*, vol. 14, no. 1, pp. 19–22, 2003.

[43] J. Lee and S. Leyffer, *Mixed Integer Nonlinear Programming*, ser. The IMA Volumes in Mathematics and its Applications. Springer, 2011. [Online]. Available: http://books.google.co.uk/books?id=jaOR-L-YCxIC

[44] M. Schlueter, “Nonlinear mixed integer based optimization technique for space applications,” Ph.D. dissertation, University of Birmingham, 2012.

[45] M. Schlueter and M. Munetomo, “MIDACO user guide,” Hokkaido University, Japan, Tech. Rep., 2013.

[46] Z. I. Berlin, “SCIP,” 2014, [Accessed: March–July 2014]. [Online]. Available: http://scip.zib.de

[47] T. Achterberg, T. Berthold, T. Koch, and K. Wolter, “Constraint integer programming: A new approach to integrate CP and MIP,” in *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems*. Springer, 2008, pp. 6–20.

[48] T. Achterberg, “SCIP: solving constraint integer programs,” *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009.

[49] ——, “Constraint integer programming,” Ph.D. dissertation, Technische Universität Berlin, 2007.

[50] M. Schlueter, “MIDACO solver,” 2009–2014, [Accessed: March–July 2014]. [Online]. Available: http://www.midaco-solver.com

[51] Python Software Foundation, “Welcome to Python.org,” 2001–2014, [Accessed: March–July 2014]. [Online]. Available: https://www.python.org

[52] SciPy developers, “SciPy.org,” 2014, [Accessed: March–July 2014]. [Online]. Available: http://www.scipy.org

[53] J. Hunter, D. Dale, E. Firing, M. Droettboom, and matplotlib development team, “matplotlib: python plotting,” 2002–2013, [Accessed: March–July 2014]. [Online]. Available: http://matplotlib.org

[54] Oracle Corporation, “MySQL :: The world’s most popular open source database,” 2014, [Accessed: March–July 2014]. [Online]. Available: http://www.mysql.com

[55] D. Freedman and P. Diaconis, “On the histogram as a density estimator: L_2 theory,” *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, vol. 57, no. 4, pp. 453–476, 1981. [Online]. Available: http://dx.doi.org/10.1007/BF01025868

[56] S. M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists*. Elsevier Science, 2009. [Online]. Available: http://books.google.com/books?id=mXP_UEiUo9wC

[57] ryanjoneil, “`python-zibopt`: Python interfaces to the ZIB Optimization Suite,” [Accessed: March–July 2014]. [Online]. Available: https://code.google.com/p/python-zibopt

TRAN, Bao-Duy

baoduy.tran@epfl.ch | tranbaoduy@gmail.com | +41 76 506 19 85
Route de la Gare 20, CH-1131 Tolochenaz, Switzerland

EDUCATION

2011 – present	**Swiss Federal Institute of Technology in Lausanne (EPFL)** – Master in Computer Science. Current GPA: **5.7/6.0**.
2003 – 2007	**Nanyang Technological University (NTU)**, Singapore – Bachelor in Computer Engineering, **First Class Honours**.
2000 – 2003	**High School for the Gifted**, National University, Vietnam – Baccalaureate with English specialisation, **Distinction**.

AWARDS & HONOURS

2007	**Scholars Award & Gold Medal** (top 3 Computer Eng. graduates) – Prestigious award for overall excellence at NTU.
2004 – 2007	**Dean's List** (top 5%, all 4 academic years) – School of Computer Engineering (NTU, Singapore).
2003 – 2007	**ASEAN Scholarship** (S$42,880) – To read Computer Engineering at NTU, Singapore.
2004	**First Runner-up Prize** – Design & Development Robotics Competition (NTU, Singapore).
2002 – 2003	English contests (Vietnam): **2x Third Prizes** (national level). **First Prize** (city level). **Gold Medal** (southern Vietnam).

PROFESSIONAL EXPERIENCE

Dec 2014 onwards [*expected*]
Google (Sydney, Australia) – **Software Engineer** (*full-time*) – Google Maps for Business [*expected*]

- Design and development of next-generation technologies that change how millions of users connect, explore and interact with information and one another, all at the the scale of the Web. Management of project priorities, deadlines and deliverables. Testing, deployment, maintenance and enhancement of software solutions.

Feb 2014 – Aug 2014
Technicolor (Rennes, France) – **Intern** – Media Computing Laboratory

- *Master's Thesis:* 'Systematic approach to multi-layer parallelisation of time-based stream aggregation under ingest constraint in the cloud'. *Technologies:* Amazon Kinesis, Java, mySQL, Python, Bash.

Sept 2013 – Nov 2013
Credit Suisse (Lausanne, Switzerland) – **Intern** – IT Development Centre

- Analysis of existing data feed system for business intelligence applications. *Technologies:* Bash, SQL, Oracle, Ctrl-M.

Sept 2011 – Aug 2013
EPFL (Lausanne, Switzerland) – **Research & Teaching Assistant** (*part-time*) – Artificial Intelligence Laboratory

- Incentive studies on Amazon Mechanical Turk; Game-with-a-purpose for sentiment annotations. *Technologies:* mySQL, Python, Bash, HTML/JavaScript, PHP, Java. *Teaching*: 'Artificial Intelligence' (bachelor's, medium: French).

Sept 2008 – Aug 2011
IBM (Singapore) **– IT Specialist** (*full-time*) – Global Technology Services

- Design and development of software solutions for integration of technologies with business applications, focusing on self-service banking and retail.
- Significant technical contribution to critical automated-teller-machine/ATM projects (deployed across South-East Asia) and customer-flow solutions (deployed to retail outlets nationwide).
- Enhancement to software engineering process for maintainability and extensibility in collaborative environment.
- Technical training workshops for new hires during their orientation month (principal instructor).
- *Technologies*: Java (SWT, JDBC, JNI, OSGi), JavaFX, C/C++ (Win32, CEN/XFS), HTML/JavaScript, JSON, 912/NDC.

Aug 2007 – July 2008
NTU (Singapore) – **Project Officer** (*full-time*) – Centre for Advanced Media Technology

- Research in graphics & visualisation via industry-collaborated projects. *Technologies:* C++, Qt, OpenSG, VRML.

Jan 2006 – June 2006
Bosch (Stuttgart, Germany) – **Software Developer** (*intern*)

- Development of applications for classification, visualisation and manipulation of production data; enhancement of monitoring and notification tools for production servers. *Technologies:* Oracle, C++ (MFC).

ACADEMIC PROJECTS

2011 – 2013
EPFL

- *Semester Project*. 'Metadata front-end for Shore-MT storage manager'. *Technologies:* Unix, C/C++, SWIG, Python.
- *Advanced Databases*. 'Distributed query & OLAP cube processing'. *Technologies:* Java, Hadoop/MapReduce, mySQL.
- *Virtual Reality*. 'Augmented-reality interactive 3D game'. *Technologies:* C++, Ogre3D, ARToolkit, BulletPhysics.

2003 – 2007
NTU

- *Bachelor's Thesis*. 'Semantic Web service composition using OWL-S'. *Technologies:* Java, XML, OWL-S.
- *Summer Project*. 'Instant message analyser'. Interactive data visualisation. *Technologies:* Java 2D, mySQL.
- *Software Engineering*. 'Bus service simulator'. Team leader for visualisation system. *Technologies:* Java 3D, VRML.
- *Inter-semester Project*. Maze traversal of logic-circuit-based robot with Java software control.

PUBLICATIONS

- B. Faltings, P. Pu, **B. D. Tran**, R. Jurca, *Incentives to Counter Bias in Human Computation*, 2nd AAAI Conference on Human Computation & Crowdsourcing, Pittsburgh, United States, Nov 2014. [*forthcoming*]
- D. N. Le, **B. D. Tran**, P. S. Tan, A. Goh, E. W. Lee, *MODiCo: A Multi-Ontology Web Service Discovery and Composition System*, 9th International Conference on Web Engineering, San Sebastian, Spain, June 2009.
- **B. D. Tran**, P. S. Tan, A. Goh, *Composing OWL-S Web Services*, 12th IEEE Conference on Emerging Technologies & Factory Automation, Patras, Greece, Sept 2007.

IT KNOWLEDGE & SKILLS

- **Java** (2D/3D, Swing, SWT, JDBC, JNI, OSGi, JavaFX, AWS), **C/C++** (MFC, Win32 API, Qt, OpenMesh, OpenSG), **Python** (matplotlib, NumPy/SciPy, mySQLdb), Visual Basic, C#/.NET. Object-oriented paradigm, **UML**. Some experience: Prolog, Scheme, assembly.
- Relational **databases** & **SQL** (mySQL, IBM DB2, MS SQL, Oracle, Access). Data processing: MapReduce/Hadoop, OLAP data cube.
- **Web**: HTML, Apache HTTP, JavaScript (MooTools, jQuery), CSS, JSON, XML/XSD, Play!, TCP/IP socket programming, PHP.
- **Linux** (principal), **Windows** (experienced); MS Office, LibreOffice, LaTeX, MATLAB; IDEs (Eclipse, JBuilder, Visual Studio), version control (Mercurial, Git, SVN), graphics (CorelDRAW, GIMP, ImageMagick, Inkscape, MeshLab), Linux command-line utilities.

LANGUAGE PROFICIENCY & PERSONAL INFO

- **Multilingual**, written & oral: **English** (*advanced,* IELTS: 8.5/9.0); **French** (*intermediate*, CEFR: B2-C1); **Vietnamese** (*native*).
- Born 8th Aug 1985, single. Vietnamese nationality. *Long-term residence:* Switzerland (2011–present), Singapore (2003–2011). *Short-term residence:* France (Feb–Aug 2014), Germany (Jan–June 2006).
- *Co-curricular activities:* Vice-President PR, IT Officer (EPFL Toastmasters), IT Officer (NTU French Society); Treasurer (EPFL GNU); Secretary (Changi-Simei Toastmasters, Singapore); Layout/Reporter (NTU Students' Union); Asian Corporate Conference (Vietnam); Sunburst Youth Camp (Singapore).
- *Hobbies & interests:* Linguistics, photography, piano, travelling, football, ping-pong, cultures.

TRAN, Bao-Duy

baoduy.tran@epfl.ch | tranbaoduy@gmail.com | +41 76 506 19 85
Route de la Gare 20, CH-1131 Tolochenaz, Suisse

FORMATION

2011 – présent	**École polytechnique fédérale de Lausanne (EPFL)**, Suisse – Master en Informatique. Moyenne actuelle : **5,7/6,0**.
2003 – 2007	**Nanyang Technological University (NTU)**, Singapour – Bachelor en Génie Informatique, **mention très bien**.
2000 – 2003	**Lycée pour les talentueux**, Université nationale, Vietnam – Baccalauréat, spécialisation en anglais, **distinction**.

PRIX & DISTINCTIONS HONORIFIQUES

2007	**Prix universitaire & Médaille d'or** (top 3 diplômés de la faculté) – Prix prestigieux pour l'excellence générale à NTU.
2004 – 2007	**Dean's List** (meilleurs étudiants – top 5%, toutes les 4 années universitaires) – Faculté de Génie Informatique de NTU.
2003 – 2007	**Bourse ASEAN** (S$42,880) – Pour faire des études universitaires en Génie Informatique à NTU, Singapour.
2004	**Deuxième prix** – Concours du design et du développement robotique (NTU, Singapour).
2002 – 2003	Concours d'anglais (Vietnam) : **2x Troisième prix** (national). **Premier prix** (ville). **Médaille d'or** (sud-Vietnam).

EXPÉRIENCES PROFESSIONNELLES

Dès déc. 2014 [*prévu*]

Google (Sydney, Australie) – **Ingénieur logiciel** (*plein temps*) – Google Maps for Business [*prévu*]

- Conception et développement des technologies de dernière génération qui influenceront la façon dont des millions d'utilisateurs connectent aux infos, les explorent et interagissent, tout à l'échelle du Web. Gestion des priorités des projets, des délais et des livrables. Testage, déploiement, maintenance et amélioration des solutions logicielles.

févr. 2014 – août 2014

Technicolor (Rennes, France) – **Stagiaire** – Laboratoire de media computing

- *Thèse de master' :* « Approche systématique à la parallélisation multi-couche d'une opération d'agrégat des streams sous le débit d'entrée limité d'un nuage informatique ». *Technologies :* Amazon Kinesis, Java, mySQL, Python, Bash.

sept. 2013 – nov. 2013

Credit Suisse (Lausanne, Suisse) – **Stagiaire** – Centre de développement IT

- Analyse d'un système data-feed existant pour l'informatique décisionnelle. *Technologies:* Bash, SQL, Oracle, Ctrl-M.

sept. 2011 – août 2013

EPFL (Lausanne, Suisse) – **Assistant de recherche/d'enseignement** (*mi-temps*) – Laboratoire d'Intelligence Artificielle

- Études des primes sur Amazon Mechanical Turk; Game-with-a-purpose pour l'annotation des sentiments. *Technologies* : mySQL, Python, Bash, HTML/JavaScript, PHP, Java (Play!). *Enseignement :* « Intelligence Artificielle » (cours bachelor ; véhicule de l'enseignement : français).

sept. 2008 – août 2011

IBM (Singapour) **– Spécialiste IT** (*plein temps*) – Global Technology Services

- Conception et développement des solutions logicielles pour l'intégration des technologies aux applications commerciales, notamment pour les banques et les points de vente en libre-service.
- D'importantes contributions techniques aux projets cruciaux des distributeurs de billets (déployés à travers l'Asie du Sud-Est) et des gestions de clientèle (déployées aux points de vente dans tout le pays).
- Amélioration des procédures génie-logiciel pour la maintenabilité et l'extensibilité dans un environnement de développement collaboratif.
- Ateliers de formation technique pour les nouveaux employés durant leur mois d'introduction (formateur principal).
- *Technologies* : Java (SWT, JDBC, JNI, OSGi), JavaFX, C/C++ (Win32, CEN/XFS), HTML/JavaScript, JSON, 912/NDC.

août 2007 – juil. 2008

NTU (Singapour) – **Responsable de projet** (*plein temps*) – Centre for Advanced Media Technology

- Recherche en graphisme/visualisation en collaboration avec l'industrie. *Technologies* : C++, Qt, OpenSG, VRML.

janv. 2006 – juin 2006

Bosch (Stuttgart, Allemagne) – **Développeur** (*stagiaire*)

- Développement des applications pour la classification, la visualisation & la manipulation des données industrielles ; amélioration des outils surveillance & notification pour les serveurs de production. *Technologies* : Oracle, C++/MFC.

PROJETS ACADÉMIQUES

2011 – 2013
EPFL

- *Projet de semestre.* « Front-end du gestionnaire de stockage Shore-MT ». *Technologies* : Unix, C/C++, SWIG, Python.
- *Bases de données avancées.* « Requêtes & cube d'OLAP répartis ». *Technologies* : Java, Hadoop/MapReduce, mySQL.
- *Réalité virtuelle.* « Jeu 3D interactif de réalité-augmentée ». *Technologies* : C++, Ogre3D, ARToolkit, BulletPhysics.

2003 – 2007
NTU

- *Thèse de bachelor.* « Composition des services Web sémantiques en OWL-S ». *Technologies* : Java, XML, OWL-S.
- *Projet d'été.* « Analyseur des messages instantanés ». Visualisation interactive. *Technologies* : Java 2D, mySQL.
- *Génie logiciel.* « Simulateur d'un service de bus ». Chef d'équipe pour la visualisation. *Technologies* : Java 3D, VRML.
- *Projet inter-semestriel.* Traversée de labyrinthe d'un robot aux circuits logiques et commandes logicielles Java.

PUBLICATIONS

- B. Faltings, P. Pu, **B. D. Tran**, R. Jurca, *Incentives to Counter Bias in Human Computation*, 2e AAAI Conference on Human Computation & Crowdsourcing, Pittsburgh, États-Unis, nov. 2014. [*à paraître*]
- D. N. Le, **B. D. Tran**, P. S. Tan, A. Goh, E. W. Lee, *MODiCo: A Multi-Ontology Web Service Discovery and Composition System*, 9e International Conference on Web Engineering, Saint-Sébastien, Espagne, juin 2009.
- **B. D. Tran**, P. S. Tan, A. Goh, *Composing OWL-S Web Services*, 12e IEEE Conference on Emerging Technologies & Factory Automation, Patras, Grèce, sept. 2007.

COMPÉTENCES INFORMATIQUES

- **Java** (2D/3D, Swing, SWT, JDBC, JNI, OSGi, JavaFX, AWS), **C/C++** (MFC, Win32 API, Qt, OpenMesh, OpenSG), **Python** (matplotlib, NumPy/SciPy, mySQLdb), Visual Basic, C#/.NET. Paradigme orienté-objet, **UML**. Petites expériences : Prolog, Scheme, assembly.
- **Bases de données** relationnelles & **SQL** (mySQL, IBM DB2, MS SQL, Oracle, Access). Traitements : MapReduce/Hadoop, OLAP.
- **Web**: HTML, Apache HTTP, JavaScript (MooTools, jQuery), CSS, JSON, XML/XSD, Play!, programmation socket TCP/IP, PHP.
- **Linux** (principal), **Windows** (expérimenté) ; MS Office, LibreOffice, LaTeX, MATLAB ; IDEs (Eclipse, JBuilder, Visual Studio), gestion de versions (Mercurial, Git, SVN), graphisme (CorelDRAW, GIMP, ImageMagick, Inkscape, MeshLab), outils commandes Linux.

COMPÉTENCES LINGUISTIQUES & INFOS PERSONNELLES

- **Multilingue**, écrit & oral : **anglais** (*avancé*, IELTS : 8,5/9,0); **français** (*intermédiaire*, CECR : B2-C1); **vietnamien** (*maternelle*).
- Né le 8 août 1985, célibataire. Nationalité vietnamienne. *Résidence à long terme :* Suisse (2011–présent), Singapour (2003–2011). *Résidence à court terme :* France (févr.–août 2014), Allemagne (janv.–juin 2006).
- *Activités extrascolaires :* Vice-Président RP, Responsable IT (EPFL Toastmasters), Responsable IT (EPFL Toastmasters, NTU Société française) ; Trésorier (EPFL GNU) ; Secrétaire (Changi-Simei Toastmasters) ; Mise-en-page/Reporter (NTU Syndicat étudiant) ; Asian Corporate Conference (Vietnam) ; Sunburst Youth Camp (Singapour).
- *Centres d'intérêts :* linguistique, photographie, piano, voyages, football, ping-pong, cultures.