Department of Computer Science
Series of Publications A
Report A-2021-6

Performance Tuning and Query Optimization for Big Data Management

Yuxing Chen

Doctoral thesis, to be presented for public examination with the permission of the Faculty of Science of the University of Helsinki, in Hall A129, Chemicum, on the 2nd of December, 2021 at 12 o'clock.

University of Helsinki
Finland

Supervisor
Jiaheng Lu, University of Helsinki, Finland

Pre-examiners
Eduardo Cunha de Almeida, Federal University of Paraná, Brazil
Uta Störl, University of Hagen, Germany

Opponent
Alberto Abelló, Polytechnic University of Catalonia, Spain

Custos
Jiaheng Lu, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911

ISSN 1238-8645
ISBN 978-951-51-7739-1 (paperback)
ISBN 978-951-51-7740-7 (PDF)
Helsinki 2021
Unigrafia

Performance Tuning and Query Optimization for Big Data Management

Yuxing Chen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
yuxing.chen@helsinki.fi

PhD Thesis, Series of Publications A, Report A-2021-6
Helsinki, December 2021, 64+120 pages
ISSN 1238-8645
ISBN 978-951-51-7739-1 (paperback)
ISBN 978-951-51-7740-7 (PDF)

Abstract

The substantial increase of real-life applications creates a large scale of ever-increasing raw heterogeneous data nowadays, correlating to the four Vs characteristics of big data: volume, variety, velocity and veracity. We discuss volume and variety challenges in this thesis. For volume, efficiently extracting valuable information and making predictions from these large-scale data are interesting to various quarters from academic researchers and industrial data scientists to customers and shareholders. For variety, much research addresses the challenges of effectively storing, collecting, processing, querying, and analyzing heterogeneous data.

This thesis pushes approaches to optimize the performance with volume and variety challenges. For volume challenges, we aim at performance tuning for big data systems. In this part, to tackle cold-start situations with no statistics for models, we leverage cost-model and triangulation to model the performance, thus leading to cost-effective prediction. For variety challenges, we aim at optimizing join queries. In this part, to fill the gap of little research on join queries with heterogeneous data (i.e., relational and tree data), we research the size bound and the worst-case optimal join algorithm with relational and tree data in contrast with only relations.

For parameter tuning, this thesis first contributes to propose a cost model for Spark workloads, which leverages Monte Carlo simulation to achieve cost-effective training. Specifically, we utilize a little part of resources and data to predict dependable performance for larger clusters and datasets even with data skewness and runtime deviations. Particularly, this work considers network and disk bounds so that it performs better with I/O-bounded workloads. Next, the thesis proposes *d*-simplexed, which models the Spark workloads by leveraging Delaunay Triangulation. Unlike other black-box ML methods, *d*-simplexed utilizes piecewise linear regression models, which can be built faster and yield better prediction. Also, *d*-simplexed is built with an adaptive sampling technique which collects few training points but achieves accurate prediction.

For join queries, this thesis studies the worst-case optimal join with relational and tree data. To this end, we first embark the study on the cross-model conjunctive query (CMCQ) with relational and tree data, and formally define the problem of CMCQ processing. We reveal that the computation of the worst-case size bound of a CMCQ is $\mathcal{NP}$-hard w.r.t query expression complexity. We then develop a worst-case optimal join algorithm called *CMJoin* to match the size bound of a CMCQ under some circumstances.

Computing Reviews (2012) Categories and Subject Descriptors:

Information systems → Data management systems → Parallel and distributed DBMSs → MapReduce-based systems
Information systems → Data management systems → Database management system engines → Database query processing
Information systems → Data management systems → Database design and models → Data model extensions → Semi-structured data

General Terms:
performance tuning, query optimization, worst-case optimal join

Additional Key Words and Phrases:
big data platforms, databases, modeling and prediction, multi-model data

Acknowledgements

It has been an amazing journey!

I would like to express my sincere gratitude to my supervisor, Professor Jiaheng Lu, for all the patience, encouragement, optimism, and wisdom through my Ph.D. studies. I would not go this far without his professional guidance and relaxing but rigorous discussion.

I would also like to thank all members of the Unified DBMS research group for the precious hang-out time that adds sweetness and warmness to the long, cold, and windy winter, as well as heated debates that inspire me with new ideas and solutions.

I appreciate the valuable time and comments from pre-examiners, Professor Eduardo Almeida and Professor Uta Störl, to improve the quality of this thesis. I also appreciate Professor Alberto Abelló and Professor Jukka Nurminen for being the opponent and faculty representative, respectively, for the public defence.

I enjoyed my time to in-depth cooperate the research with Peter Goetsch, Mohammad Hoque, Chen Chen, Valter Uotila, and Professor Herodotos Herodotou. I would also like to thank the Doctoral Programme in Computer Science (DoCS), Huawei, Academy of Finland, and Oracle for their generously financial support to my research work. Special thanks to Research Coordinator Pirjo Moen and all other colleagues from the HR team and IT team, who have made my work as convenient as possible.

Finally, I want to thank my family and friends for their unconditional love and support all the way.

It is the beginning of a new journey.

Shenzhen, November 2021
Yuxing Chen

Contents

List of Publications

This thesis is based on the following original publications, none of which have been included in any other theses. The publications are referred to in the text as Papers I–VII, and have been attached to the end of this thesis.

I. Herodotos Herodotou, Yuxing Chen, and Jiaheng Lu. A survey on automatic parameter tuning for big data processing systems. *ACM Computing Survey*, 53(2):43:1–43:37, 2020.

   The author of this thesis participated in surveying the solutions and wrote the paper.

II. Yuxing Chen, Jiaheng Lu, Chen Chen, Mohammad Hoque, and Sasu Tarkoma. Cost-effective resource provisioning for spark workloads. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 2477–2480, 2019.

   The author of this thesis participated in designing the solution, implemented proposed algorithms, conducted experiments, and wrote the paper.

III. Yuxing Chen, Mohammad A. Hoque, Pengfei Xu, Jiaheng Lu, and Sasu Tarkoma. Cost-effective resource provision prediction and recommendation for spark workloads. *Submitted to Journal of Distributed and Parallel Databases (DPD)*, 2021.

   The author of this thesis participated in designing the solution, implemented proposed algorithms, conducted experiments, and wrote the paper.

IV. Yuxing Chen, Peter Goetsch, Mohammad A Hoque, Jiaheng Lu, and Sasu Tarkoma. d-simplexed: Adaptive delaunay triangulation for performance modeling and prediction on big data analytics. *Preprint, IEEE Transactions on Big Data*, 2019.

The author of this thesis designed the parameter recommendation algorithm. He implemented proposed methods, conducted experiments, and wrote the paper.

V. Yuxing Chen. Worst case optimal joins on relational and XML data. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1833–1835, 2018.

The author of this thesis participated in designing the solution, implemented proposed algorithms, conducted experiments, and wrote the paper.

VI. Yuxing Chen and Jiaheng Lu. Worst-case optimal algorithms for cross-model conjunctive queries. *Submitted to Conference on Database Systems for Advanced Applications (DASFAA)*, 2021.

The author of this thesis participated in designing the solution, implemented proposed algorithms, conducted experiments, and wrote the paper.

Besides the above papers which contributed to this thesis, the author of this thesis also participated in the following tutorial paper:

VII. Jiaheng Lu, Yuxing Chen, Herodotos Herodotou, and Shivnath Babu. Speedup your analytics: Automatic parameter tuning for databases and big data systems. *The Proceedings of the VLDB Endowment (PVLDB)*, 12(12):1970–1973, 2019

The author of this thesis participated in surveying the solutions and wrote the paper.

Chapter 1

Introduction

The substantial increase of real-life applications like E-commerce, the Internet of Things, and healthcare is producing a huge scale of ever-increasing raw data every now and then [59]. Such big data challenges correlate to the four V's characteristics of big data: volume, variety, velocity, and veracity. In this thesis, we focus on two aspects of them: volume and variety.

In the perspective of *volume*, extracting valuable information and predictions from these large-scale raw data is interesting to people from academic researchers and industrial data scientists to customers and shareholders. Yet, it is challenging to store, collect, process, query, and analyze large-scale heterogeneous data cost-effectively. To address these challenges, big data processing systems like Hadoop[1], Spark[2] and multi-model databases like ArangoDB[3], Orientdb[4] have come to the fore. To further achieve timely and cost-effective performance for big data processing systems and databases in a cluster, tuning proper parameter values and optimizing queries are encouraging yet backbreaking for not only industrial practitioners but also academical researchers.

Understanding an ample space of configurable parameters for systems seems an impossible mission, as these parameters control a spectrum of system processes, ranging from higher-level scheduling and management to low-level threading and caching [44]. The complexity of these parameters may come from the huge parameter configuration space, system internals and scales, as well as the shortage of data statistics and workload profiles. An improper set of parameter values

[1]https://hadoop.apache.org/
[2]https://spark.apache.org/
[3]https://www.arangodb.com/
[4]https://www.orientdb.org/

may deteriorate system performance and stability processing [25, 45], for example, due to some bottlenecks during the parallel. Also, the proliferation of the Cloud, Platform-as-a-Service, and pay-as-you-go service require tenants to make a cost-effective decision on requesting resource parameter values [37, 79]. Too little resource allocation may lead to a job failure, while allocating all the reaming resources for a job may not always lead to the best performance with respect to resource utilization [85] or execution time [40].

This thesis first aims to address the system parameter tuning challenges by proposing a low-cost training method of cold-start scenarios in the case of Spark platform. Specifically, it focuses on answering the following research questions (RQs):

RQ1. Given a workload and a dataset in the cold start scenarios, how do we build the "white" performance prediction model with low training cost?

RQ2. Given a workload and a dataset in the cold start scenarios, how do we build the "black" performance prediction model with low training cost?

In the perspective of *variety*, the large-scale raw data that comes from different applications consists of various forms of data, especially including structural data like relational, key/value, and graph data, and semi-structural like JSON and XML documents. Unlike traditional databases that can process only one form of data, a multi-model database is powerful and efficient to manage and process various forms of data in one engine and meets the expanding requirements for performance and scalability [60, 63]. Yet, it is challenging to process and optimize queries that are in multiple data forms, especially join queries, which are applied by data integration in data lake [41], query processing in multi-model databases [61] or in polystores [27], query processing in computational linguistics [83, 86, 92], etc.

Previous work applied naive or no optimizations on (relational and tree) cross-model conjunctive query (CMCQs) (more formal definition in Chapter 2). There exist two kinds of solutions. The first is to use one query to retrieve the result from the system without changing the nature of the model [66, 90]. The second is to encode and retrieve the tree data into a relational engine [2, 11, 69, 93]. Even though the second solution accelerates tree pattern matching, they may both suffer from generating significant unnecessary intermediate results, as they naively combine data from different models or even output results separately for different models. Not to mention, these solutions or optimizations did not consider worst-case optimality.

This thesis aims to address the worst-case optimality problems by proposing the cross-model conjunctive query (CMCQ), which integrates both relational

conjunctive query and conjunctive tree pattern together. Specifically, it focuses on answering the following research questions:

RQ3. Given relations and tree pattern queries, how do we find the worst-case size bound of the result?

RQ4. Given relations and tree pattern query, how do we design worst-case optimal join algorithm based on the size bound?

This thesis contributes several efficient approaches to answer the above research questions, which are briefly explained in the following.

1.1 Overview

The research questions consist of two problems. One is the parameter tuning for big data platforms, and the other is the worst-case optimal join under the context with multiple data forms. We briefly describe the contribution in this part and refer readers to the details in the following chapters.

For parameter tuning, there exist numerous pieces of excellent related research, targeting the three main challenges that of the vast parameter space, scalable and complex systems, and lacking statistics for system, workloads, and data. Papers I [44] and VII [59] summarized these challenges and surveyed the state-of-the-art work into 6 categories (described in Chapter 2). These categories are described as rule-based, cost modeling, simulation-based, experiment-driven, machine learning, adaptive methods, distinguishing by their modeling techniques, number of covered parameters, system understanding, training logs, etc. (described in Table 2.1). Most of the papers proposed one of the methods and addressed one of the challenges by the methods.

In this thesis, we focus on the third challenge of parameter tuning that happened in an environment with a new workload. It is usually referred to as a cold-start situation. In such a situation, the accumulated training overhead of running large-scale data is prohibitive. By the time of competing for learning the profiles of workload, the workload may be no longer valid due to the variation of workloads or updated services [56]. Worse, they are often given the assumption of the fixed input data and resource, which sometimes should be individualized for workloads [29, 76]. As for the first and second challenges of the large parameter space and complex system internals, we narrow down the problem to consider the resource parameters only, as now the exponential using of cloud, the proper

resource parameters are the key for organizations to obtain cost-effectiveness. We tried two perspectives to advance the research. Firstly, from the perspective of a white model for RQ1, Paper II [20] and III [18] proposed a detailed cost model for Spark performance predictions. Then they leveraged Monte Carlo (MC) simulation to achieve low-cost training, thus cost-effective prediction and recommendation. Then, from the perspective of a black model for RQ2, Paper IV [17] proposed d-simplexed, which models the performance topology for Spark workloads. It utilized a d-dimensional mesh through *Delaunay Triangulation* by modeling a set of d parameters. To further lower the cost of training for an accurate model, it tried to collect fewer training points by an adaptive sampling technique.

For the worst-case optimal join problem, its initial idea comes from the AGM size bound [5] for relational conjunctive joins. AGM models the join problem into the fractional edge cover problem, and computes the size bound by the linear programming in polynomial time. Based on this size bound, several optimal algorithms have been developed (e.g., *NPRR* [67], *LeapFrog* [78], *Joen* [21]) under the worst-case scenarios. However, these methods are pure with relational engines, which cannot be directly applied to the new context with multiple forms of data.

To process multiple types of data format efficiently, this thesis embarks the study on the cross-model conjunctive query (CMCQ) and formally defines the problem of CMCQ processing, which integrates both relational conjunctive query and conjunctive tree pattern together. Firstly for RQ3, Paper V [16] proposed a multi-model processing framework for relational and semi-structured data (i.e., XML and JSON). Paper VI [19] contributes to compute the size bound of a CMCQ and reveals that the computation of the worst-case size bound of a CMCQ is $\mathcal{NP}$-hard w.r.t query expression complexity. Then for RQ4, Paper VI also developed a worst-case optimal join algorithm called *CMJoin* to match the size bound of a CMCQ under some circumstances.

The contents of Papers I–VII are summarised as follows.

Paper I: A survey on automatic parameter tuning for big data processing systems.

This paper investigated existing methods on parameter tuning for both batch data processing systems and stream data processing systems, and categorize them into six approaches: rule-based, cost modeling, simulation-based, experiment-driven, machine learning, and adaptive. We analyze the pros and cons for each approach and bring out some open research problems for automatic parameter tuning. We introduce these techniques in Chapter 2.

Paper II: Cost-effective resource provisioning for spark workloads

This paper proposes a cost model for Spark performance predictions, which utilizes Monte Carlo (MC) simulation to achieve low-cost training. Specifically, MC uses a little part of resources and data to make a dependable prediction for larger clusters and datasets, even if some data samples exhibit skewness and runtime deviations because repeated simulations yield reliable profiling characteristics. Compared to Ernest, this work considers network and disk bounds so that it performs better with I/O-bounded workloads. We introduce these techniques in Chapter 3.

Paper III: Cost-effective resource provision prediction and recommendation for spark workloads.

This paper proposes a cost model for Spark performance predictions, which utilizes Monte Carlo (MC) simulation to achieve low-cost training. Specifically, MC uses a little part of resources and data to make a dependable prediction for larger clusters and datasets, even if some data samples exhibit skewness and runtime deviations because repeated simulations yield reliable profiling characteristics. Compared to Ernest, this work considers network and disk bounds so that it performs better with I/O-bounded workloads. Note this paper is an extension of Paper II. We introduce these techniques in Chapter 3.

Paper IV: d-simplexed: Adaptive delaunay triangulation for performance modeling and prediction on big data analytics.

d-simplexed is a performance prediction framework for tuning Spark workloads. Unlike other black-box ML methods, *d*-simplexed leverages a *d*-dimensional mesh via Delaunay Triangulation by modeling a set of d parameters. The key idea is that piece-wise linear regression models could be built faster and yield better predictions than complex models like Gaussian processing or Neural Networks. To further speed up model construction time, *d*-simplexed proposes an adaptive sampling technique which collects a few training points but achieving accurate prediction. We introduce these techniques in Chapter 4.

Paper V: Worst case optimal joins on relational and XML data

To process multiple types of data format efficiently, it proposed a multi-model processing framework for relational and semi-structured data (i.e., XML), and sketched a worst-case optimal join algorithm for it. We further discuss the details in Chapter 2.

Paper VI: Worst-case optimal algorithms for cross-model conjunctive queries

This paper embarks on the study on the cross-model conjunctive query (CMCQ) and formally defines the problem of CMCQ processing, which integrates both relational conjunctive query and conjunctive tree pattern together. It reveals that the computation of the worst-case size bound of a CMCQ is $\mathcal{NP}$-hard w.r.t query expression complexity. We develop a worst-case optimal join algorithm called *CMJoin* to match the size bound of a CMCQ under some circumstances. We further discuss the details in Chapter 5.

Paper VII: Speedup Your Analytics: Automatic Parameter Tuning for Databases and Big Data Systems

This paper reviews existing approaches on automatic parameter tuning for databases, Hadoop, and Spark, which we classify into six categories: rule-based, cost modeling, simulation-based, experiment-driven, machine learning, and adaptive tuning. The paper describes the foundations of different automatic parameter tuning algorithms and present pros and cons of each approach. The paper also highlight real-world applications and systems, and identify research challenges for handling cloud services, resource heterogeneity, and real-time analytics. We further discuss the details in Chapter 2.

1.2 Outline

The rest of this thesis is organized as follows. Chapter 2 defines the research problems and introduces some preliminaries. Chapter 3 presents a white-box cost model to cost-effectively provision resources for Spark workloads, and Chapter 4 builds a black-box performance model by Delaunay Triangulation with an adaptive sampling for fast convergence. As for worst-case optimal join, Chapter 5 investigates the worst-case size bound for relations and tree pattern queries, then designs the worst-case optimal join algorithm accordingly. Last, Chapter 6 concludes this thesis and elucidates future work.

Chapter 2

Preliminary

This chapter gives the formal definition of the research problems solved in this thesis. It also establishes the preliminary approach used in the upcoming chapters. We will split this chapter into two parts. The first part is the big data management preliminary, which will introduce the parameter tuning problem. The second part is the multi-model management preliminary, which will present the relational worst-case optimal join problem.

2.1 Parameter Tuning

This section gives the formal definition of the research problems for parameter tuning. It also establishes preliminary models used in the upcoming chapters.

2.1.1 Problem statement

A job J, which executes on a big data platforms, is of the form $J = \langle w, i, c, p \rangle$, where w denotes the workload of the job, i denotes the input data properties, c denotes the cluster resources, and p denotes the set of parameters used by w. Let p_i denote the i-th parameter, taking values from a finite domain $D(p_i)$. The *configuration space* $\mathbb{S}$ is the Cartesian product of the domains, i.e., $\mathbb{S} = D(p_1) \times \cdots \times D(p_n)$ and $c = \langle p_1, ..., p_n \rangle$. The performance of job J is described as $perf = F(w, i, c, p)$, where *perf* denotes a performance metric like throughput, latency, resource efficiency, etc.

We formally define the *parameter tuning problem* in the following: Given a workload w to execute input data i over cluster resources c, find the optimal

configuration parameter values p^* that maximizes F over the configuration space $\mathbb{S}$:

$$p^* = \underset{c \in \mathbb{S}}{\operatorname{argmax}} F(w, i, c, p) \tag{2.1}$$

The performance function F is usually unknown or partially known from past measurements, while several experiments with Hadoop, Spark, and Storm, have shown it to be non-convex and multi-modal [37, 45, 49]. Moreover, finding an optimal solution in such a setting is NP-hard [84].

2.1.2 Tuning approaches

Parameter tuning techniques are promising for big data platform optimization. However, it remains challenging to seek optimal values for performance. Firstly, big data platforms like Hadoop and Spark have more than 200 configuration parameters [9, 52, 74]. Even as challenging, the parameters are not independent from each other [45, 49]. Secondly, the system becomes complex with scalable nodes, different hardware and software [57], as well as parallelism execution [37] Thirdly, most users are faced with the cold-start situation without any statistics for data, workloads, and systems [22, 44].

Much research has targeted one or two of the above challenges with a wide range of methods to achieve high performance. The related work can be classified into six approaches: (1) rule-based, (2) cost modeling, (3) simulation-based, (4) experiment-driven, (5) machine learning, and (6) adaptive. These categories are summarized by many features of different approaches, such as modeling method, number of parameters involved, the level of requirement for historical logs and input statistics, etc., which are presented in Table 2.1. Interestingly, these features are addressing the three challenges mentioned above. For more details of the approaches, we refer readers to Papers I [44] and VII [59].

2.2 Worst-case optimal join

This section gives the formal definition of the research problems for relations and tree data. It also establishes preliminary relational join and data structures used in the upcoming chapters.

Table 2.1: Feature comparison among the six parameter tuning approaches.

Feature	(1)	(2)	(3)	(4)	(5)	(6)
modeling technique	rules	cost model	simulation	search algorithms	ML models	mixed
# of parameters involved	few	some	some	many	many	some
System understanding	strong	strong	strong	light	no	strong
Need for history logs	no	light	light	strong	strong	light
Need for data input stats	no	light	light	no	strong	light
Real tests to run	no	some	no	yes	yes	yes
Time to build model	efficient	efficient	medium	slow	slow	medium
# of metrics predicted	few	few	some	few	many	some
Prediction accuracy	low	medium	medium	medium	high	medium
workload adaptive	adaptive	light	light	no	no	adaptive
system adaptive	no	no	light	no	adaptive	light

2.2.1 Relational size bound

We review the size bound for the relational algebra, which is originally proposed by Asterias, Grohe, and Marx (AGM) [5]. AGM modeled the size bound by the fractional edge cover problem and computed the size bound by linear programming (LP). Formally, given a relational schema $\mathbb{R}$, for every table $R \in \mathbb{R}$, let A_R be the set of attributes of $\mathbb{R}$ and $\mathcal{A}{=}\cup_R A_R$. Then the worst-case size bound is precisely the optimal solution for the following LP:

$$\begin{aligned} \text{maximize} \quad & \Sigma_r^{\mathcal{A}} x_r \\ \text{subject to} \quad & \Sigma_r^{A_R} x_r \leq 1 \quad for\ all\ R \in \mathbb{R} \\ & 0 \leq x_r \leq 1 \quad for\ all\ r \in \mathcal{A} \end{aligned} \tag{2.2}$$

Let ρ be the optimal solution of the above LP, and the size bound is N^ρ, where N is the upper bound of each table. The AGM bound can be proved as a special case of the discrete version of a well-known inequality in geometry: the Loomis-Whitney inequality [58]. Interested readers may refer to the details of the proof in the AGM paper [5].

Example 2.1 *Consider a typical triangle conjunctive query:* $R_0(a,b,c){:=}R_1(a,b) \bowtie R_2(b,c) \bowtie R_3(a,c)$. *Then we, based on three relations, can generate the three LP inequalities, i.e.,* $x_a + x_b \leq 1$, $x_b + x_c \leq 1$, $x_a + x_c \leq 1$. *Therefore, the maximize value of* $\Sigma_r^{\mathcal{A}} x_r = x_a + x_b + x_c$ *is* $\frac{3}{2}$, *meaning the query* $R_0(a,b,c)$ *yields the worst-case size bound* $\mathcal{O}(N^{\frac{3}{2}})$.

2.2.2 Relational worst case optimal join

Given a query Q along with a set of size constraints for each input table, how do we compute the worst-case size bound of the output results for Q. This is an interesting problem, because deriving the bound is not only useful for analyzing the runtime of the algorithm, but also instrumental in designing the worst-case optimal algorithms, which can guarantee that the intermediate result size is no more than the worst-case bound. Recently, Grohe and Marx [39] and Atserias, Grohe, and Marx [5] model conjunctive joins into the fractional edge cover problem, allowing us to compute the worst-case size bound by linear programming. Based on this worst-case bound, several worst-case optimal algorithms have been proposed (e.g., *NPRR* [67], *LeapFrog* [78], *Joen* [21]). For example, Ngo et al. [67] construct the first algorithm whose running time is worst case optimal for all natural join queries. Veldhuizen [78] proposes an optimal algorithm called *LeapFrog* which is efficient in practice to implement. Ciucanu et al. [21] propose an optimal algorithm *Joen*, which joins each attribute at a time via an improved tree representation. There also exist research works on considering functional dependencies (FDs) for size bound. The initiated study with FDs is from Gottlob, Lee, Valiant, and Valiant (GLVV) [36], which introduces an upper bound called GLVV-bound based on a linear program on polymatroids. The follow-up study by Gogacz et al. [32] provide a precise characterization of the worst-case bound with entropy theory. Khamis et al. [54] provide a worst-case optimality for any query where the GLVV bound is tight. See an excellent survey on the development of worst-case bound theory [68].

2.3 Chapter summary

This chapter introduced the preliminaries for the optimization of big data management and multi-model data management for the following chapters. For optimization of big data management, we introduced a parameter tuning problem, while for multi-model data managment, we introduced the size bound and related worst-case join algorithms.

Chapter 3

Resource Provisioning by Cost Modeling

This chapter provides an overview of the results of Papers II and III that solve RQ1. Following a formal definition of RQ1, it introduces a cost model with a simulation framework that predicts the cost-effective resource provision. Finally, this chapter presents empirical experimentation results to highlight the performance improvement of the proposed method.

3.1 Motivation

There exist excellent previous works [44, 59] addressing parameter tuning challenges by leveraging machine learning techniques, including Decision Tree (C5.0) regression [81], Logistic regression [50], Gradient Boosting regression [42], and Reinforcement Learning [91] or by search-based algorithms, including bound-and-search [94], Random Forest [7], and Genetic Algorithm [89]. Each method, having its own superiority, targets one of the partial problems of the three challenges discussed in Chapter 2. Not surprisingly, the success of these black-box approaches requires sufficient quantity and good quality of historical log data, which are usually in short and need to be collected from many runs [65, 88]. Thus it is a costly process [55, 91] when these techniques are for large-scale dynamic workloads. The reason is that the repeated run of workload for large-scale data is prohibitive. Worse, by the time of finishing training, the training may not be useful due to the variation of workloads [56]. Another concern is that, they often assume fixed workload, data, and resource, which is not individualized enough for workloads [29, 76]. There are various cost models (e.g., [80, 82]) proposed as

well. Wang et al. [82] and Ernest [80] run workloads with smaller input data size and predicted the performance for the larger dataset by using mathematical cost functions. However, these machine learning and cost models are limited by the network performance and disk I/O. They do not recommend the resource parameter values either.

This section addresses the challenges in cost-effective resource allocation for Spark jobs with no start statistics. We particularly choose the white cost model against costly black machine learning training. Our cost model is superior to catch the I/O bottleneck when data scale. The detail of the approach will be described in the following section.

3.2 Problem statement and preliminary

We introduce the problem statement, the performance metric for evaluation, as well as the cost model for prediction and recommendation in this part.

3.2.1 Problem statement

We describe the problem in the following:

- **Input:** (i) a set of workloads/jobs $\mathbb{J}$; (ii) available nodes, vcore, and memory; (iii) timely or cost-effective requirements defined by users.
- **Output:** for each $\mathcal{J} \in \mathbb{J}$ with data size D, we recommend number of vcores V; amount of memory M.
- **Objective:** high performance parameter settings V and M.

For different scenarios, it often configures different parameter settings to achieve the high performance of different metrics. We describe the performance metric in the following.

3.2.2 Performance metric

The execution running time is a primitive but essential performance metric for workload evaluation. In this part, we introduce two of the most important metrics, namely resource time (e.g., used in [38]) and throughput (e.g., used in [3]). However, the recommendation model is not limited to extend with other time-related metrics.

Resource time, T_R. Workload executing time reveals the speed of the computation. However, it is not sufficient to reveal the resource consumption [38]. Today, more and more the pay-as-you-use cloud service, e.g., Amazon on-demand pricing is widely being used. Given the workload execution time T and total amount assigned resource of vcore V and memory M, we define memory time T_M $(= M \times T)$ and vcore time T_V $(= V \times T)$. The resource time T_R then can be represented as follows:

$$T_R = T_V + \theta T_M \tag{3.1}$$

where θ describes the renting price ratio between per GB memory and per vcore.

Choosing T_R as the performance metric is two-fold. Firstly, T_R can quantify actual used resource and its corresponding cost, as well as the power consumption [85]. Secondly, it can explain the fairness [31] in a multi-tenant cluster. Therefore, we explore the vcore and memory parameter values to achieve minimum T_R. Given a set of vcore and memory parameter values (V,M) and its running time performance T, v_{rec} and m_{rec} achieve minimum resource time, meaning that:

$$v_{rec}, m_{rec} = arg \min_{v \in V, m \in M} T_R(v, m) \tag{3.2}$$

By default, we use this resource time as the metric. And to achieve cost-effectiveness, we aim at maximizing this metric.

3.3 Approach

Figure 3.1 shows the general description of our modeling and prediction processes as well as the data flow between the processes. To begin with, we sample and separate the original input data according to the data size and available resources, i.e., vcores and memory. With sampled data, we run the workloads with few resources to learn statistics for prediction. We first ensure the memory requirement for input data, which can satisfy (i) avoiding out-of-memory errors for non-caching jobs or (ii) data fitting into the memory for caching jobs. Then, we construct the mathematical cost model to analyze sampled data, which records the profiles of jobs, to predict the performance with a larger data size and cluster. Finally, we can aim at optimizing a performance metric, e.g., timely runtime, cost-effective allocation, and highest throughput.

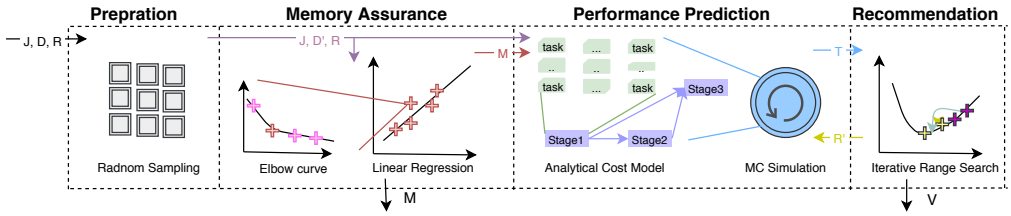


Figure 3.1: Approach visualization and processing data flow. The process consists of random sampling and MC simulation, inflection creation and linear regression, analytical cost model, iterative range search. [Abbreviation: Job (J), Data (D), Available Resource (R), Sampled Data (D'), Allocated Memory (M), Execution Time (T), Shrank Resource (R'), Allocated Vcores (V)].

3.3.1 Simulation

Simulation of a workload is executing the workload with a small portion of data to estimate or learn the performance with original-size data. We introduce Independent Bernoulli Sampling [70] (with replacement sampling) to evenly select the data samples, assuming independent and identical distribution.

It is critical to decide how many samples are needed, as too few samples are not sufficient to learn the profile of the characteristics, and they are also the challenges of:

- Data skewness: the runtime of a job can be different by two data samples with the same data size.
- Runtime deviation: the runtime of a job can be different by the same data and resource configuration.

To mitigate the problem, Monte Carlo (MC) Simulation (e.g., [10]) is introduced to iteratively train model with samples till a convergent one. The training cost accumulates the resource time of the sample runs. The training cost of a job can be described as the ratio of resource times between simulations and actual run as follows:

$$C_{training} = \frac{\sum_{i=1}^{n} T_{r_i}}{T_R} \tag{3.3}$$

Where T_{r_i} indicates the i^{th} round of resource time in simulation; n is the number of executions for simulations; T_R is the resource time for the actual run. It is worth mentioning that there exists a trade-off between the training cost and prediction

accuracy. A confidence interval (CI) is determined to stop the simulation or training.

3.3.2 Memory assurance

Designing a proper sampling experiment can reveal the actual need for memory. Allocating sufficient memory notably helps for data to fit in while extra memory often does not help, and large memory may cause long GC time [6]. The simulation run of samples for memory can be depicted as an elbow curve [53]. The elbow point in the elbow curve is interesting to us, as it is the derivative of the curve function with a huge jump. This elbow point can be the best resource time representative. Let P express distinct sampling probabilities $\{p_1, p_2, ...p_i, ..., p_n\}$, where p_i is the i-th probability for sample. For each p_i's simulation and by the elbow point, we obtained assured memory m_i and $M = \{m_1, m_2, ...m_i, ..., m_n\}$. Then by a regression model, we estimate the assured memory $\hat{M}$ for the original size data when being provided P and M. In this section, we leverage the *linear regression* model for this prediction, as it practically performs well for the benchmark jobs.

3.3.3 Performance prediction

In this subsection, we introduce the cost model for job performance prediction. A spark job consists of many stages, and each stage consists of multiple parallel tasks. For a stage, we record the task time by the longest execution time, as it stands for the completion time. The cost model considers not only the computation time but also the input, output, and shuffle time. The computation time is mainly restricted by the number of vcores, whereas others are mainly restricted by the network bandwidth and disk read/write performance.

Computation speed for different jobs may vary for the prediction model, as the computation complexity varies in jobs, while for the reading/writing time for input, output, and shuffle are fixed for the model. We collect the simulation logs from the history server using Spark REST APIs [1] to learn the computing speed. As for reading and writing speeds, we firstly record the speed by testing the complete runtime by *Linux dd*, then carefully revised them by calculating the runtime of some stressing jobs (e.g., sort for shuffling throughput as sort is shuffle-intensive). These values cannot come from simulations, as the sample runs often do not reach the bottleneck.

3.3.4 Parameter recommendation

In this subsection, we implement a search algorithm, called *iterativeRS* to find the near-optimal vcore configuration. The algorithm narrows down the range to the (near-) optimal value. The core idea is to predict the performance of given resource range. Then from that range, we can locate a smaller range for better performance. We shrank the range and repeat the prediction till we reach near-optimal values with some stopping criteria. For simplicity, we assume our problem is to find the local minimizer. The maximizing problem of finding throughput can be transformed into a minimizing problem. We describe the algorithm as follows:

1. For the given resource range, partition the space into multiple choices.
2. Run simulation to predict performance by the parameter values in this range choice.
3. Predict the local optimal range from this prediction.
4. Narrow down the range, and if it is satisfy terminated criteria, we stop, otherwise update the resource range and go back to Step 1.

We next show the evaluation result of the performance prediction and parameter recommendation in the following.

3.4 Evaluation

In this section, we first evaluate the prediction model, and then evaluate the parameter recommendation model based on performance metrics derived from the prediction model.

3.4.1 Experimental setup

We experimented on the performance of HiBench workloads in a Spark cluster managed by Yarn. We set up 12 benchmark workloads in 3 different categories: micro, machine learning, and SQL. The cluster has 10 resource nodes, each with 32 GB of RAM and 2 Intel Xeon E5540 (with 8 cores) 2.53 GHz CPUs. In the view of Yarn resource manager, we have 10 worker nodes, and each node has 16 vcores (by *hyperthreading*) and 32 GB memory. Also, we set up five data nodes for HDFS storage. All the nodes have two bonded 10 $Gbit/s$ network interfaces. The versions for Hadoop, Spark, and HiBench [47] are 2.7.3, 2.1.1 and 7.1 respectively.

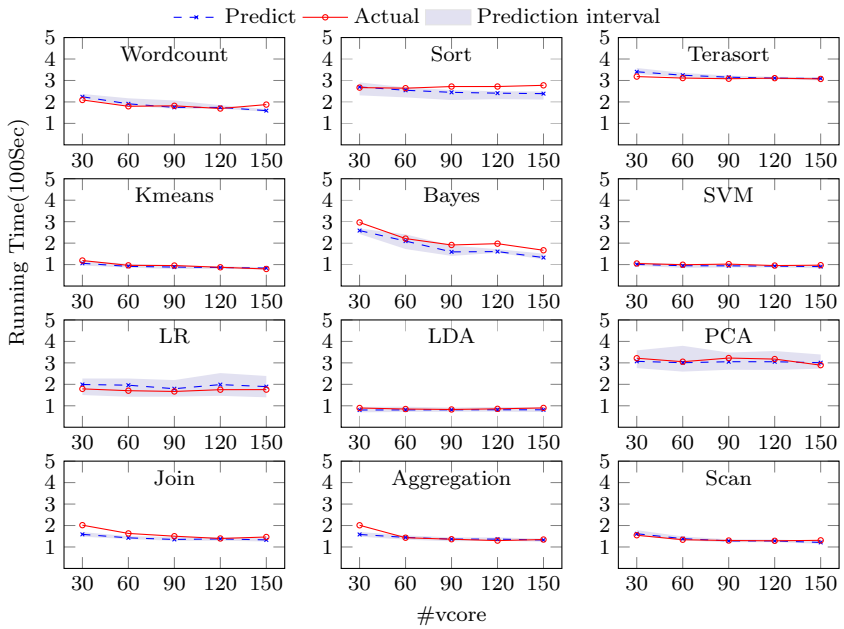


Figure 3.2: Performance prediction of 12 workloads. Predicted vcores (dashed curve) vs. actual needed vcores (solid curve).

We maximized the executor size in our evaluation, as fewer executors in one node often yield a better performance [77].

3.4.2 Performance prediction

Figure 3.2 presents the comparison result of benchmarking workloads between the predictions and actual runs. The dashed curves stand for the predicted results and the solid curves stand for the actual run results. We show the prediction interval by drawing the shadow with the minimum and maximum boundary. In general, the cost-model can predict the curves of the workloads. For example in Bayes workload, as we increase the vcores for sample simulation, the performance has been improved while the actual run did it similarly. Unexpectedly, Sort and Terasort did not work well against the actual run. The reason is that these two workloads have I/O bottlenecks, so adding more computing resources does not help the workload to facilitate the I/O waiting. To sum up these inaccuracies, the observation is that overhead is underestimated, as the I/O context switching

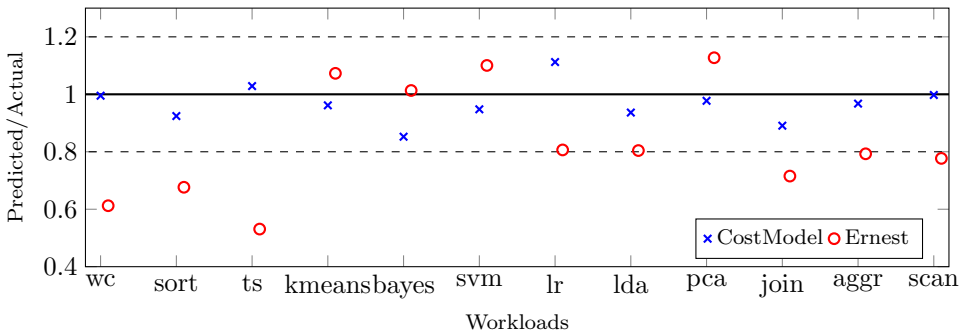


Figure 3.3: The prediction precision of execution time. The closer to 1, the better.

or waiting may affect the I/O itself or even the computation. Besides, there exist some prediction results with a huge shadow interval, for example, LR and PCA. This implies that the skewness of data distribution or performance deviation is more severe due to some randomness in the workload process. Therefore, LR and PCA need more simulation costs to achieve a stable prediction.

Prediction precision. To look closer to the precision, running time is used as precision metric to compare a state-of-the-art cost model Ernest [80]. Figure 3.3 illustrates the prediction precision for the provided benchmarking jobs. Overall, the precision error is under 20% for each job, and the average precision error is under 5%. In a production environment, this error would not influence the decision-making on determining appropriate parameter settings. Overall for all machine learning workloads, we perform as well as Ernest prediction. However, Ernest does not predict accurately for Mirco and SQL workloads, as it does not consider the network and disk bottlenecks as data and resource scales.

3.4.3 Parameter recommendation

Table 3.1 shows the performance speedup via our parameter recommendations. We calculate the speedup in T_R by dividing the performance runtime for the baseline configuration (with all resources assigned) by the performance runtime for the recommended one. We can obtain up to 657% T_R performance speedup by some benchmark workloads. The recommendation not only satisfies the deadline criteria required by users but also achieves remarkable saving in resource time.

Training cost Table 3.1 presents the training cost ratio for the tested 12 benchmark workloads. When setting 80% CI, the accumulated training costs

Table 3.1: Evaluation of benchmark jobs concerning resource time and training cost ratio for resource parameter recommendation.

Type	Job	Data size	Reccomend	T_R speedup	CI=80%	CI=90%
Micro	Wordcount	91.8GB	60V18G	217%	4.11%	4.47%
	Sort	9.55GB	30V18G	492%	1.99%	2.37%
	TeraSort	9.55GB	30V18G	536%	1.86%	2.37%
ML	Kmeans	11.2GB	60V36G	229%	5.89%	6.51%
	Bayesian	14.0GB	90V54G	161%	7.91%	16.89%
	SVM	26.8GB	30V36G	440%	5.30%	5.30%
	LR	15.2GB	30V18G	644%	34.83%	54.77%
	LDA	246.2MB	30V18G	657%	7.6%	11.05%
	PCA	30.7MB	30V18G	499%	9.36%	41.80%
SQL	Join	17.3GB	30V18G	360%	4.99%	5.22%
	Aggregation	17.3GB	30V18G	335%	5.33%	6.06%
	Scan	17.3GB	30V18G	423%	5.74%	6.83%

for tested workloads are within 10% except for LR, as LR runtime deviates dramatically because of the data skewness and the subtle effects of the system. Likewise, when given 90% CI, the costs are within 20% except for LR and PCA. Therefore, the average costs are 7.92% and 13.64% comparing to one-time actual workload runs for CI=80% and CI=90%, respectively. It is worth mentioning that the storage cost is negligible. The reason is that the *REST API* collects *INFO*-level logs which will be further processed and trimmed and far less than the input data size.

3.5 Chapter summary

This chapter introduced Paper II and III, in which we proposed a cost-effective prediction to answer RQ1. We presented the simulation framework and detailed cost model to learn the workload statistics by training the samples. Furthermore, we used these trained models to recommend cost-effective resource provision, thus achieving optimized resource utilization and cost.

Chapter 4

Performance Modeling by Delaunay Triangulation

This chapter provides an overview of the results of Paper IV that solves RQ2. Following a formal definition of RQ2, it introduces a Delaunay Triangulation model that models the performance topology. Finally, this chapter presents empirical experimentation results to highlight the performance prediction accuracy of proposed methods.

4.1 Motivation

Tuning often aims at improving workload performance by searching one optimizing parameter value. Unluckily, tuning from one goal to another goal usually needs to retrain the model. The optimal parameter value for running time is not necessarily cost-effective [20, 80]. In this work, we model the performance topography of the whole feature configuration space. Differently from tuning, modeling is able to make decisions on the best resource allocation [15, 75], lowest resource consumption [38], highest data throughput [3], shortest completion runtime [73], etc. It would be challenging but promising to model a performance topology with a few but decisive parameter values. Such a surface can help for further optimal decisions, such as the cost-effectiveness (choosing best number of nodes [80]) in clouds like `Amazon` "pay-as-you-go" services[1], or cost-saving resource allocation but deadline-satisfied in a cluster [15].

[1] `https://aws.amazon.com/ec2/pricing/on-demand/`

Many excellent works for performance modeling and tuning in different platforms have been researched, e.g., Apache Hadoop (e.g., [46, 73]), Apache Spark (e.g., [43, 80]), Database systems (e.g., [3, 26]). Particularly, Alvaro [43], OtterTune [3], and CDBTune [91] build multiple regressors to tune performance. However, they build these models to maximize only one objective, such as running time, meaning that it does not fit well when turning to other objectives such as throughput, thus ending up retraining the models. Also, these state-of-the-art regressors either face a deterioration accuracy by their over-fitting (Gaussian Process [3] and multi-layer Neural Network [91]) or achieve unsatisfactory accuracy due to under-fitting (Linear regression [43], Ernest [80]).

Facing these challenges, we apply a novel efficient regressor with an adaptive sampling to mitigate the training cost. We aim at building a bootstrap performance surface model, which accurately predicts unknown parameter values in whole space. Taking inspiration from the field of Computational Geometry, we model a $d+1$-dimensional mesh (e.g., [30]) by d critical parameters with Delaunay Triangulation (DT) [24]. DT has a wide range of practical usages like curved surface modeling [14] in computational geometry and path planning in automated driving [4] in computer graphics. Then by the constructed mesh, we can predict the performance in the unknown parameter value area.

4.2 Problem statement and preliminary

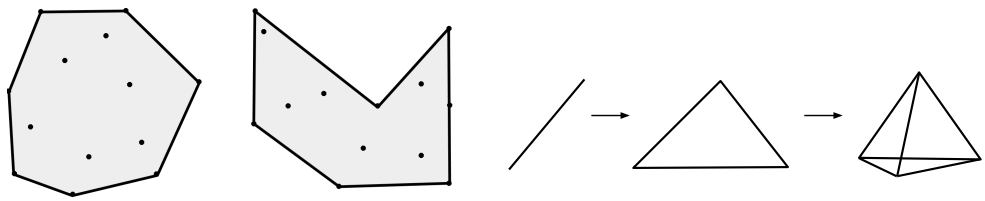
In this part, we introduce several basic terms and concepts on Delaunay Triangulation (DT) in the field of Computational Geometry. Then we give our problem definition for this chapter.

4.2.1 Delaunay Triangulation primitives

We present a basic idea of some terms that relate to DT as follows:

- **Convex Region** [64] is a region that, for each pair of points, every point on the straight line segment that joins the pair of points is also within the region. Figure 4.1a shows an instance, depicting the convex and non-convex regions.

- **Convex Set** [64] illustrates the points in a convex region. In convex geometry, it is a subset of an affine space that is closed under convex combinations [13].



(a) (b)

Figure 4.1: (a) On the left is the convex hull constructing a convex region and on the right is a non-convex hull constructing a non-convex region. (b) from the 1-simplex (line), 2-simplex (triangle) to the 3-simplex (tetrahedron).

- **Convex Hull** [8] of a set of points is the smallest convex set that contains the points.

- **Hyperplane** is the generalization of a plane in three dimensions to higher dimensions, i.e., any d subspace in $\mathbb{R}^{d+1}$.

- **Simplex** is the generalization of a triangle to different dimensions. In d dimensions, the concept of a triangle becomes a d-simplex. Figure 4.1b depicts an example of a $2d$ triangle to a 2-simplex, a $3d$ tetrahedron to a 3-simplex, and so on.

By introducing these concepts, our model is constructed by d parameters of Spark job into a d-dimensional space by utilizing DT to split the space into many interconnected d-simplexes, thus constructing a $d+1$-hyperplane via these parameter sets.

4.2.2 Problem statement

In general, we want to predict the runtime performance of a workload by its varied parameter configurations when given historical data points. We develop a performance model based on these known historical data points and to predict performance of the unknown parameter values. Formally, given n samples of historical logs, a parameter configuration space F, a prediction model $M(\cdot)$ finds the predicted running time $\hat{T}$ for F:

$$\hat{T} = M(\mathcal{S}, F) \tag{4.1}$$

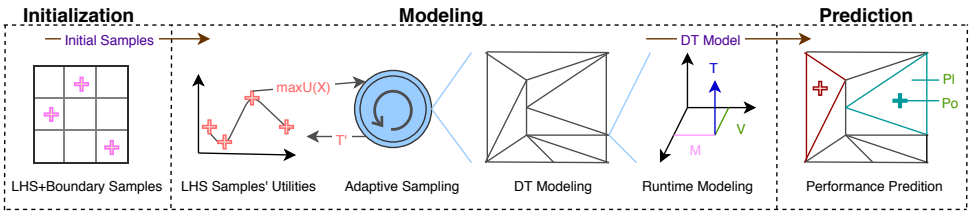


Figure 4.2: d-Simplexed overall architecture. Denote: Utility ($U(X)$) of samples X, Runtime (T), Predicted time (T'), Memory (M), vcore (V), Value point to be estimated ($Po = \langle f_1, f_2 \rangle$), Plane to be estimated ($Pl = \beta_1 x_1 + \beta_2 x_2 + \beta_3$).

where $\mathcal{S}$ is the set of history logs $\{\langle F_i, T_i \rangle | 1 \leq i \leq n\}$, and each configuration setting $F_i = \{f_1, f_2, ..., f_d\}$ consists of d features[2] and its runtime is T_i. For empirical experiments of our method, we utilize the Mean Absolute Percentage Error (MAPE) as our metric, which is the average errors in percentage for all predicted runtimes ($\hat{T}$) and actual runtimes (T):

$$MAPE = \frac{100\%}{l} \sum_{i=1}^{l} |\frac{T_i - \hat{T}_i}{T_i}| \tag{4.2}$$

where l is the number of configuration settings to be run.

4.3 Approach

To address the problem, we propose d-Simplexed by utilizing the Delaunay Triangulation (DT) model and an adaptive sampling technique, to predict a given configuration setting and reduce training samples and the cost, respectively. Figure 4.2 shows our framework at a high level. The key components are adaptive sampling, DT modeling, projection, and performance prediction, which will be introduced with more details in the following.

4.3.1 Delaunay Triangulation

The main part of d-Simplexed is to model the performance mesh with given parameters. A Delaunay Triangulation(DT) [24] is introduced here to model the performance by a set of parameters. Given P a set of discrete points in a plane,

[2]Features are not limited to system parameters (e.g., vcore and memory). The job or input data information (e.g., data size) can be a feature.

generic triangulation $T(P)$ holds that no point p in P is within the region of any triangle in $T(P)$, and DT tries to maximize the minimum angles of all the triangles in $T(P)$. DT holds the following advantages:

- It favors equilateral triangles instead of long and skinny triangles, thus interpolating values better (predicting model is more fit) [8, 23].
- It is extendable to high dimensions. The model is extendable when adding more features to the prediction.

We next discuss the key process to implement the model to our problem:

- **Triangulation:** Given a set of d features (parameters) $\{f_1, f_2, ..., f_d\}$, e.g., f_1= {16 GB, 4 vcores}, the Delaunay Triangulation model is built in $\mathbb{R}^d$ space;
- **Projection:** For every d-simplex from the DT model, we map running-times as $(d + 1)$ points to calculate the hyperplanes;
- **Prediction:** Given a new configuration setting F, DT locates the corresponding simplex and predicts the performance.

In the following, we will describe more details and implementations of above processes.

4.3.2 Modeling

In this subsection, we describe the modeling of the workload statistics by DT.

Triangulation. With d features selected, a d-dimension DT mesh is constructed. In 2 dimensions, DT is formed by 2-simplex, which is $2d$ triangles, while in 3 dimensions, DT is formed by 3-simplex, which is a $3d$ tetrahedron. In higher dimensions, the construction works in exactly the same way except for the complexity for visualization. In DT, the whole space is split into regions. And we build the model locally and separately, comparing to fit a single model to the whole space. To efficiently implement DT, we apply the Quickhull [8] algorithm. The runtime of algorithm Quickhull is in $\Theta(n \log n)$ and in worst-case $O(n^2)$. Since the chosen points may not be efficient for the model training, we will introduce adaptive sampling later to speed up the training.

Projection. Given features d with runtime, we can build a multivariate linear function representing a hyperplane in $\mathbb{R}^{d+1}$. When given a new point with d

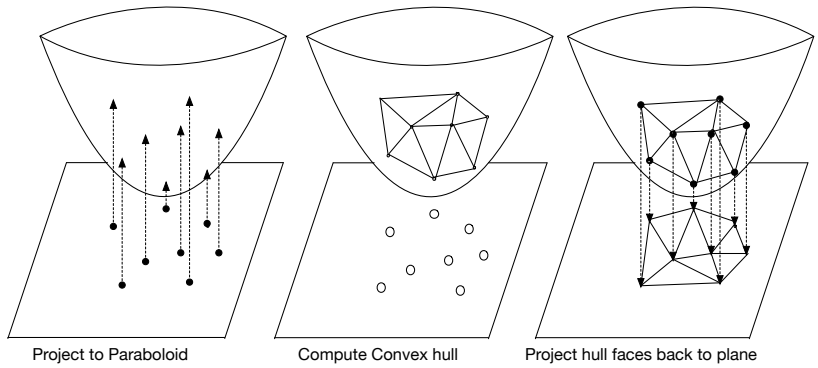


Figure 4.3: The example of computing the convex hull and project the convex hull back by a parabola $(x^2 + y^2)$.

features, we locate the correct d-simplex to predict for runtime in $d+1$ dimension. The multivariate linear function is piece-wise, making it fast and easy to build yet still accurate to predict. DT first split the space into several simplexes. Then for each simplex, the hyperplane is constructed when mapping to the runtime dimension. For example, in 2 dimensions, i.e. $\langle f_1, f_2 \rangle$, DT yields 2-simplexes (triangles), with each containing three data points. And for each point $\langle f_1, f_2 \rangle$, we map the runtime to the third dimension ($runtime(f_1, f_2)$), then compute the hyperplane by these three points. For general higher dimensions with d features, we lift it into $\mathbb{R}^{d+1}$ by including the runtime metric. Example 4.1 depicts an example to illustrate the modeling.

Example 4.1 *In Figure 4.3, we show the simplex and hyperplane relation in 2-dimension (x, y) by a parabola $(x, y, x^2 + y^2)$. We calculate the convex hull in $\mathbb{R}^{2+1}$ space by lifting the points to a parabola, then project the convex hull back into $\mathbb{R}^2$ space.*

4.3.3 Prediction

After we built the DT model, when given a new configuration $F = \langle f_1, f_2, ..., f_d \rangle$, we can predict the runtime by the DT model. Specifically, we first locate the simplex belonging to F. Then we can calculate the values of the hyperplane for the simplex. By this hyperplane, we can predict the runtime for F. The mathematical form of a hyperplane with d dimensions is $\beta_0 = \sum_{i=1}^{d+1} \beta_i x_i$. Given $d+1$ known configurations and the runtimes, we find the parameter values $\{\beta_0, ..., \beta_{d+1}\}$.

$h(\cdot)$ is a prediction function to return runtime (i.e., x_{d+1}) when given new F in Equation 4.3. When the simplex's hyperplane is calculated, every time a new configuration F feeds to Equation 4.3, we can predict the runtime.

$$T = x_{d+1} = h(x_1, ..., x_d) = \frac{\beta_0 - \sum_{i=1}^{d} \beta_i x_i}{\beta_{d+1}} \tag{4.3}$$

Note that Points lying outside the convex hull are difficult to predict as these values cannot be extrapolated by a hyperplane. To address these, we can set a boundary for space at the beginning.

4.3.4 Adaptive sampling

When the feature space has a highly symmetrical distribution, the model may perform unsatisfactorily. To address such skewness, we introduce an adaptive sampling method. The baselines are random sampling and grid sampling. The random sampling selects samples randomly without replacement. The grid sampling splits the space into uniform grids and selects samples with an equal distance from each neighbor. The baselines may over-sample in some unimportant regions, which barely help to improve the models. We present a feedback-driven method to pick samples that differ from each other, thus avoiding over-fitting.

To make sure each hyperplane estimates the monotonic surface, we greedily explore the sparse region with less known parameter values, while to narrow down errors, we greedily select the region with dramatic performance changes (defined in Equation 4.4). Specifically, the sampling technique is two steps. To begin with, we initialize the model with some seed samples, which can be random or one-time Latin Hypercube Sampling (LHS) sampling [48]. Then, samples are continuously added to improve the model. The samples that are added to the model are selected based on a *utility* function. The utility function is to calculate the distance between the estimated points and their hyperplane. In general, the higher the utility value, the larger distance to its hyperplane, meaning a higher potential point to improve the model. Let X be the samples from LHS sampling $\mathcal{D}_{LHS}$ and n samples $\mathcal{S}$ with d features, we can obtain the predicted performance running time $\hat{T^{(i)}}$ by the sample $X^{(i)}$ and sample's hyperplane $\mathcal{S}' \subseteq \mathcal{S}$ with h sample $\langle F_k^{(i)}, T(F_k^{(i)})\rangle, 1 < k < h$. Then, $U(X^{(i)})$ of sample $X^{(i)}$ can be defined in the following:

$$U(X^{(i)}) = \frac{1}{h}\sum_{k=1}^{h}(\frac{1}{d+1}(\sum_{j=1}^{d}(f_{k,j}^{(i)} - X_j^{(i)})^2 + (T(F_k^{(i)}) - \hat{T^{(i)}})^2)) \tag{4.4}$$

where h denote the number of points to construct the hyperplane. The sampling technique iteratively execute until we meet a stopping criteria.

By a utility function, now we can rank all the sample based on their utility values. The largest value point is obtained:

$$F_{(n+1)} = arg \max_{X \in \mathcal{D}_{LHS}} U(X) \tag{4.5}$$

We iteratively select new sampling points until a stopping criterion is met (e.g., MAPE$\leq$ 5%).

4.4 Evaluation

In this section, we first evaluate the performance model, then evaluate the adaptive sampling.

4.4.1 Experiment setup

We set up Apache Spark v2.1.0 and Hadoop v2.8.1 with YARN and HDFS. The cluster has 10 nodes, each with 32 *GB* RAM, 2 Intel Xeon E5540 each with 4 cores, making 16 vcores available with hyperthreading. The state-of-the-art approaches are the Multivariate Linear Regression (LR) [43], Decision Tree Regression (DTR) [43], Gaussian Process (GP) [3], Ernest [80] and Multilayer Neural Network (NN) [91]. The baseline sampling techniques are random and grid samplings. The workloads are HiBench [47] benchmarking workloads (WordCount, PageRank, Kmeans clustering, Bayesian classification, and SQL Join) and complex synthetic surface to simulate diverse workloads for Spark or any other systems.

Figure 4.4 presents the model accuracy of all workloads with 2% training samples. Overall for the evaluated workloads, *d-Simplexed* performs better than the LR, GP, DTR, and NN. *d-Simplexed* obtains within 5% MAPE for all given workloads except for Bayesian 11.281%. In complex synthetic workloads SingleWave and MultiWave, *d-Simplexed* obtains 9.576% and 20.723% MAPE, respectively.

Among all baselines, DTR is quite satisfactory due to its piece-wise regression, which is similar to *d-Simplexed*. GP and NN generally outperform LR for Kmeans, Bayesian and synthetic workloads, as the runtime performance changes abruptly (usually non-linear changing) with different parameter values. This is because GP and NN excel in non-linear curves fitting comparing to LR. While in WordCount and Join workloads with flatter performance, LR outperforms GP.

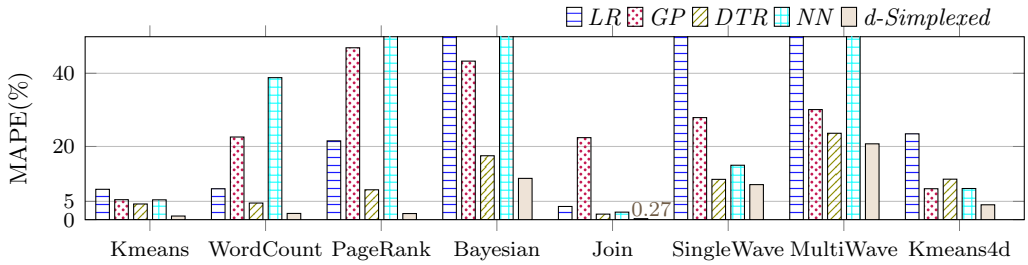


Figure 4.4: Evaluating MAPE (The lower the better.) for d-Simplexed agianst LR, GP, DTR, and NN. We train 2% of samples. We cut more than 50% MAPE results for better illustration.

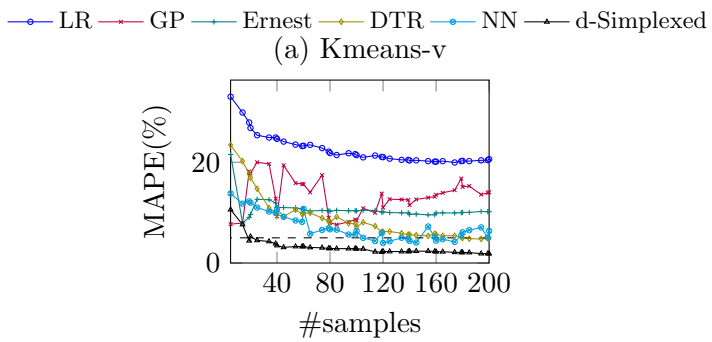


Figure 4.5: Evaluation of the *d*-Simplexed against baselines for the Kmeans workload. *d*-Simplexed achieves the best MAPE with 200 training samples.

4.4.2 Model evaluation

The first set of experiments evaluates *d*-Simplexed with adaptive sampling, comparing to other baseline methods, namely LR, GP, Ernest, DTR, and NN. As Ernest only works for data and vcore scale, we start with these two features. Figure 4.5 presents the empirical accuracy by the Kmeans workload. *d*-Simplexed outperforms six baseline models with regard to MAPE. Ernest outperforms LR as Ernest also considers logarithm terms in the cost function which is more flexible to capture the workload. GP and NN flip around as samples increase, which is a typical over-fitting. As Ernest only discusses vcore feature, we will not include it in the rest evaluation.

We now add a memory feature in the following evaluations. Table 4.1 presents the result against baseline approaches concerning MAPE. We only used 1% of the

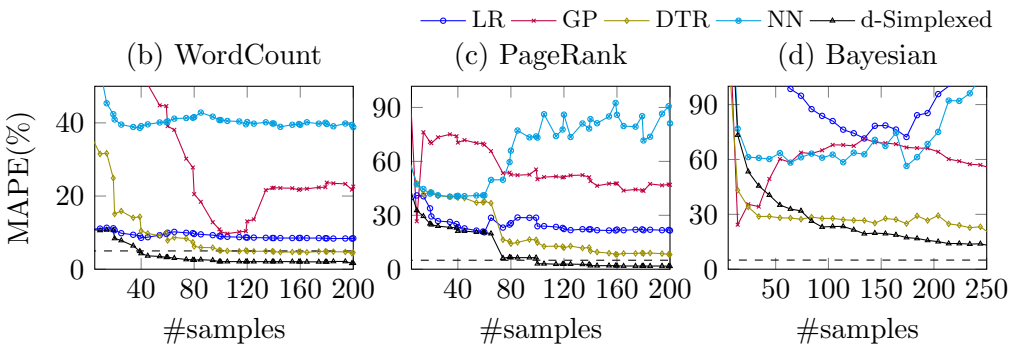


Figure 4.6: Evaluation of the d-Simplexed against baselines for Kmeans, WordCount, PageRank, and Bayesian workloads. d-Simplexed obtains the best MAPE.

Table 4.1: Evaluations of Kmeans, WordCount, PageRank, Bayesian, and Join workloads with LR, GP, DTR, NN, and DT with random sampling (DT-RD), DT with grid sampling (DT-GD), and DT with adaptive sampling (d-Simplexed). The evaluation metric is the MAPE (%) with 1% of training samples. Parameter space: start value - step size - end value.

Workload	**Data**	**Config space**	LR	**GP**	**DTR**	**NN**	**DT-RD**	**DT-GD**	**d-Simp-**
Kmeans	80G	M:40-2-240 V:60-1-160	9.379	28.017	5.645	27.218	2.91	2.51	**1.58**
WordCount	80G	M:40-2-240 V:60-1-160	8.93	10.256	5.085	40.772	2.086	4.665	**2.071**
PageRank	80G	M:40-2-240 V:60-1-160	23.914	50.076	14.765	73.218	4.354	10.763	**3.053**
Bayesian	14G	M:10-1-120 V:10-1-120	85.36	66.111	26.646	68.481	20.091	345.045	**15.768**
Join	17.3G	M:10-1-120 V:10-1-120	3.278	20.357	1.833	2.391	2.674	0.954	**0.367**

training samples. d-Simplexed outperforms other approaches in terms of MAPE. d-Simplexed is under 5% MAPE for all workloads except 15.768% MAPE for the Bayesian workload.

Figure 4.6 illustrates more evaluation results. Particularly, LR predicted worse in Kmeans and Bayesian jobs. This is because a linear model has trouble in learning a non-linear performance curve. In contrast, GP and NN predicted well in these jobs. However, they have an over-fitting problem when data points are increased adding into the model. So, GP and NN did not work well for workloads with flatter performance like WordCount and PageRank. DTR separates space piece-wisely, leading to better MAPE. Also, DTR computes an average value to form a regressor rather than a linear regressor in d-Simplexed. So, DTR poorly has

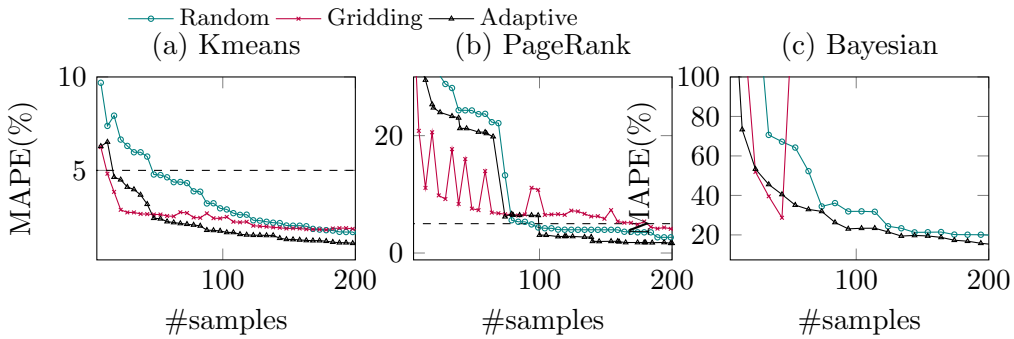


Figure 4.7: Evaluation of adaptive sampling to random and griding sampling. The dashed line is the 5% MAPE line.

some limitations in local predicting. Unluckily, *d*-Simplexed does not consistently outperform in the starting phase. The reason is that non-linear regressors capture the curves better in such cases.

As we increase the samples, *d*-Simplexed outperforms baselines. The reason is that *d*-Simplexed fit very well locally and does not affect globally as data point added. Also, by our proposed adaptive sampling, we speed up the sample selections, thus leading to better accuracy.

4.4.3 Sampling evaluation

The second set of experiments evaluates the performance of DT with adaptive sampling, comparing to random and grid sampling. Table 4.1 and Figure 4.7 provide the experimental results of *d*-Simplexed compared to DT with other sampling techniques, i.e., DT with random sampling (DT-RD) and DT with grid sampling (DT-GD) in terms of MAPE. *d*-Simplexed can perform better than baseline samplings as we increase the sample points. DT-RD is not satisfied, especially for the starting phase. The reason is that the randomness of picking samples makes the model unstable. DT-GD is better than DT-RD, as it uniformly picks samples for the model. In contrast, *d*-Simplexed with adaptive sampling picks the samples that maximize the possibility to the model. This behavior is especially pronounced, for example in Bayesian workload, where *d*-Simplexed focuses on the critical region of the topology with a steep performance change.

The third set of experiments evaluates *d*-Simplexed with more complex synthetic workloads and more input features. We refer readers to Paper IV [17] for more details.

4.5 Chapter summary

This chapter introduced Paper VI, in which we proposed a performance modeling technique to answer RQ2. We proposed the *d*-Simplexed framework consisting of a performance model formed by the Delaunay Triangulation (DT) model and an adaptive sampling method. The DT model can model the performance topology accurately, while the adaptive sampling helps speed up the training process.

Chapter 5

Worst-case Optimal Join for Relations and Trees

This chapter gives an overview of the results of Papers V and VI that solve RQ3 and RQ4. Following a formal definition of RQ3, we find out it is not a polynomial work to compute multi-model data with relational and tree data. Finally for RQ4, this chapter presents a worst-case optimal algorithm to compute the multi-model join queries under some circumstances.

5.1 Background and preliminary

This section introduces the motivation and provides some preliminaries for the following sections.

5.1.1 Motivation

Traditional relational databases research heavily for conjunctive (join) queries as they are the most common analytical tools. The size bound for a join query is the cardinality of the join result in the worst case. And the corresponding worst-case optimal algorithms guarantee that the runtime can be completed in linear of the size bound. In this section, we study the Cross-Model Conjunctive Queries (CMCQs) over relational and tree data. The size bound for relational conjunctive query has been widely researched [5, 33, 35]. Yet the size bound for tree pattern conjunctive queries remains challenging and unkown, let alone the mixed quires of both forms. Besides this theoretical interest, it also has several applications that have highly related to this topic. The application ranges from

data integration in data lake [41], query processing in multi-model databases [61] or in polystores [27], query processing in computational linguistics [83, 86, 92], etc.

5.1.2 Problem statement

Let $\mathbb{R}$ be a database schema, R_1,...,R_n are relation names in $\mathbb{R}$. A rule-based conjunctive query over $\mathbb{R}$ is an expression of the form $R_0(u_0) \leftarrow R_1(u_1) \wedge R_2(u_2) \wedge ... \wedge R_n(u_n)$, where $n \geq 0$, R_0 is a relation not in $\mathbb{R}$. $u_0, u_1, ..., u_n$ are free tuples, i.e. may use either variables or constants. Each variable occurring in u_0 must also occur at least once in u_1,...,u_n. The semantics of the above query can be explained with a conjunctive calculus query $\{e_1, ..., e_m | \exists x_1, ..., x_k (R_1(u_1) \wedge R_2(u_2) \wedge ... \wedge R_n(u_n))\}$, where $x_1, ..., x_k$ are all the variables in the relations occurring in the body and not the head, and $e_1, ..., e_m$ are all variables occurring in the head u_0.

Let $\mathcal{T}$ be a tree pattern, where there exist two binary axis relations: `Child` and `Descendant` axes which are defined in the normal way [35]. In general, a cross-model conjunctive query contains three components: the relational expression τ_1, the tree expression τ_2 and the cross-model label expression τ_3, as follows:

$$\tau_1 := \exists r_1, ..., r_k, R_1(u_1) \wedge R_2(u_2) \wedge ... \wedge R_n(u_n) \tag{5.1}$$

where r_1,...,r_k, are all the variables in the relations R_1,...,R_n.

$$\tau_2 := \exists t_1, ..., t_k, Child(v_1) \wedge ... Descendant(v_n) \tag{5.2}$$

where $t_1, ..., t_k$ are all the node variables occurring in v_i ($i \geq 1$) and each v_i is a binary tuple (t_{i_1}, t_{i_2}).

Let Σ denote a labeling alphabet. Given any node $t \in T$ and $T \in \mathcal{T}$, $label(t, s)$ means that the label of node t is $s \in \Sigma$. The label relations bridge the expressions of relations and trees together by the equivalence between the labels of tree nodes and the values of relational data items.

$$\tau_3 := \exists r_1, ..., r_k, t_1, ..., t_k, Label_1(t_{i_1}, r_{j_1}) \wedge ... \wedge Label_n(t_{i_n}, r_{j_n}) \tag{5.3}$$

Definition 1 (Cross-model cunjunctive query (CMCQ)) *Given a query Q with relations $\mathcal{R}$ and tree patterns $\mathcal{T}$, $\mathcal{R}$ has the form of Equation 5.1, $\mathcal{T}$ has the form of Equation 5.2, and the matching of $\mathcal{R}$ and $\mathcal{T}$ has the form of Equation 5.3, then a join result of all attributes $\{e_1, ..., e_m\}$ of CMCQ is a calculus form of:*

$$\{e_1, ..., e_m | \tau_1 \wedge \tau_2 \wedge \tau_3\}. \tag{5.4}$$

(a) Child axe (b) Tree instance (c) Descendant axe (d) Tree instance

Figure 5.1: Tree conjunctive queries with (a) Child axes and (c) Descendant axes, and their worst-case instance tree (b) for query (a) and tree (d) for query (c).

5.2 Size bound for relational and tree data

Given a cross-model conjunctive query (CMCQ) Q with relation and tree pattern to match, we first assume the maximum relational table size and the tree node for each attribute is at most N. We now discuss how to find the maximum cardinality of the result for Q. To begin with, we investigate two basic but important tree pattern queries. We will discuss the descendant axes and child axes in a tree pattern, and how they are equivalent to the inequalities for linear programming.

5.2.1 Introducing tree data into join

Let us start with a single child axe and a single descendant axe. The worst-case result size of a child axe pattern match is $\mathcal{O}(N)$, as each child node is a one-to-one matching parent node. While the worst-case result of a descendant axe pattern match is $\mathcal{O}(N^2)$, as each descendant node can be the descendant of the ancestor node.

Example 5.1 *Given the basic patterns of a single child axe in Figure 5.1a and a single descendant Figure 5.1c, Figure 5.1b shows one of the possible ways to construct the instance tree that obtains $\mathcal{O}(N)$ for the child axe, and Figure 5.1d shows one of the possible way to construct the instance tree that obtains $\mathcal{O}(N^2)$ for the descendant axe.*

Recall that in Section 2.2, each relation is equivalent to an inequality in linear programming, meaning that each relation is bounded to $\mathcal{O}(N)$. So intuitively, the combined attributes of a child axe are equivalent to an inequality, while a descendant axe does not have any restrictions on that.

Definition 2 (PC-path inequality) *Let P be a PC path in the tree pattern* **T** *and A_P denote the set of labels of nodes in P. The PC-path inequality of P is defined as $\Sigma_r^{A_P} x_r \leq 1$.*

A PC-path is equivalent to an inequality which is easy to prove as each child has only one parent and grandparent and so on. So the result size of a PC-path pattern matches $\mathcal{O}(N)$. For example, Figure 5.1a presents a child axe which is equivalent to a PC-path inequality $x_a + x_b \leq 1$.

The worst-case size bound for a query Q with only child axes is the solution for the following LP problem, which includes both relational inequalities and PC-path inequalities:

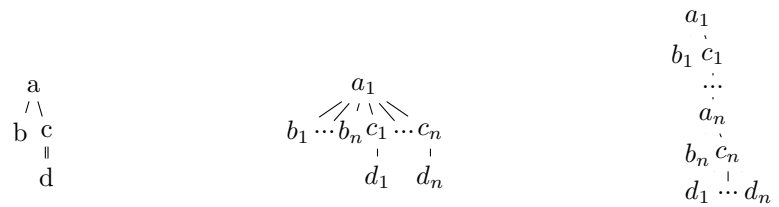
$$\begin{aligned} \text{maximize} \quad & \Sigma_r x_r \\ \text{subject to} \quad & \Sigma_r^{A_R} x_r \leq 1 \quad for\ all\ R \in \mathbf{R} \\ & \Sigma_r^{A_P} x_r \leq 1 \quad for\ all\ P \in \mathbf{T} \\ & 0 \leq x_r \leq 1 \quad for\ all\ r \in A_R \cup A_T. \end{aligned} \tag{5.5}$$

Example 5.2 *Given a CMCQ Q with relation $R(b, c)$, and a tree pattern T=$a[b]/c$, R is equivalent to $x_b + x_c \leq 1$ and T is with two PC paths a/b and a/c, which are equivalent to $x_a + x_b \leq 1$ and $x_a + x_c \leq 1$, respectively. Then the maximum value of $x_a + x_b + x_c$ is $\frac{3}{2}$, meaning the size bound of Q is $\mathcal{O}(N^{\frac{3}{2}})$.*

At this point, we can achieve an upper bound for CMCQ with both child and descendant axes, by considering a PC-path an inequality and a descendant axe without any restriction. However, this is not a tight bound as for some cases, a descendant axe can not yield $\mathcal{O}(N^2)$ results when some branches are considered no restriction to each other.

Example 5.3 *Consider the query in Figure 5.2a with only a tree pattern. The corresponding PC-path inequalities are $x_a + x_b \leq 1$, $x_a + x_c \leq 1$. Thus, the maximum value of $x_a + x_b + x_c + x_d$ is 3 when x_b=x_c=x_d=1 and x_a=0 (one of the possible solution). However, it is infeasible to construct a tree instance with $\Theta(N^3)$ matching result. In fact, the tight upper bound is only $\mathcal{O}(N^2)$. Figure 5.2b and Figure 5.2c show two instances of trees in the worst case situation.*

In this case, when we obtain $\mathcal{O}(N^2)$ result for b and c in Figure 5.2b, c and d can not yield $\mathcal{O}(N^2)$ result any more, meaning the c and d is no more equivalent to no constraint for descendant axe, and vice versa. So bound seems to be two alternatives: (i) $x_a + x_b \leq 1$, $x_a + x_c + x_d \leq 1$, and (ii) $x_a + x_b + x_c \leq 1$, $x_d \leq 1$, responding to two instance trees. They obtain the same size bound $\mathcal{O}(N^2)$. These two different alternatives are meaningful to compute the size bound with more complex cases. For example, give relation $R_1(b, c, d)$ (corresponding to $x_b + x_c + x_d \leq 1$), then with inequalities (ii), it leads to the maximum bound



(a) Tree query (b) Tree instance 1 (c) Tree instance 2

Figure 5.2: Tree conjunctive query (a) and its alternative worst-case instance tree (b) and tree (c).

2, *while with the inequalities (i) it can obtain only* $\frac{3}{2}$. *For another example with relation* $R_3(b, d)$ *and* $R_4(a, c, d)$ *(corresponding to* $x_b + x_d \leq 1$ *and* $x_a + x_c + x_d \leq 1$*), this time with inequalities (i) is a winner with maximum value* 2, *compared to* $\frac{3}{2}$ *for the inequalities* (ii).

Given a CMCQ with relations, child axes, and descendant axes, the above example hints at a possible method to construct inequalities to evaluate the size bound. For one query, it may construct multiple alternatives of inequalities to make sure about the bound. Unlike in the AGM bound for only relations, which needs only polynomial complexity, the evaluation of the size bound for a CMCQ is $\mathcal{NP}$-hard regarding query complexity.

Theorem 5.1 (NP-hardness) *The query complexity of the worst-case bound evaluation of databases for a cross-model conjunctive query is* $\mathcal{NP}$*-hard.*

The central proof is to polynomially reduce the 1-IN-3SAT problem [71] to our problem. See details in Paper VI for the proof.

Remark Note that this complexity is with respect to the size of the query. It should not be confused with the data complexity of the query answering algorithm later, which has polynomial complexity regarding data complexity. Considering a practical size of a query is limited, from the point of view of applications, the above theorem mainly makes it of theoretical interest. In contrast to relational conjunctive queries that are *tractable* with respect to query complexity, this paper contributes to demonstrating the theoretical complexity gap due to the occurrence of tree patterns in a conjunctive query.

5.2.2 Recursive conversion and split

This subsection introduces a concrete algorithm that finds the size bound for a CMCQ. The central idea is to eliminate all descendant axis relations in the tree pattern recursively by two operations, called *conversion* and *split*. We terminate when the final tree pattern remains with only the child axes which can be mapped into LP inequalities. We then obtain the size bound by choosing the optimal solution for all constructed LP inequalities.

Definition 3 (Conversion and split operations) *Let T be a tree pattern and α be a descendant axis between x and y nodes in T. Assume that x is the parent of y.*

- ***Conversion**: $T \mid \alpha$ denotes an operation to convert the descendant axis α to the child axis between x and y.*
- ***Split**: $T \parallel \alpha$ denotes an operation to remove α from T and thereby T is split into two subtrees. It is important to note that this split operation must be adjoined with one or multiple compensation inequalities defined below.*

Definition 4 (Compensation inequalities) *With respect to the split operation in a tree pattern T, assume that node x is split from node y. Let P denote the root-to-x path. For each root-to-leaf path P' that does not contain x, let $\mathbf{A}$ denote all labels in P and P', then we generate a compensation inequality of P': $\Sigma x_r \leq 1$ for all labels $r \in \mathbf{A}$.*

We first explain compensation by the tree pattern in Figure 5.2a. When the ancestor node c is split from descendant node d, a compensation inequality is generated: $x_a + x_b + x_c \leq 1$. The reason is when the split of node c and d is happening in the worst-case situation in Figure 5.2c, meaning every instance node of c matches every instance node d. In this case, every instance node of c has one parent node of a and this node of a must have at least one child b node. So the result size of a, b, and c is bound to $\mathcal{O}(N)$, i.e., $x_a + x_b + x_c \leq 1$.

The central idea of computing the size bound for the tree pattern is to determine if we split the AD pair and generate compensation inequalities or we consider AD pair as an inequality. These alternatives generate *canonical suites* which are equivalent to LP inequalities. When there exist multiple tree patterns, we can straightforwardly combine them by a dummy root.

Definition 5 (Suites and canonical suites) *A suite is a triple tuple (*$\mathbb{R}$,$\mathbb{C}$,$\mathbb{T}$*), where* $\mathbb{R}$ *denotes relations,* $\mathbb{C}$ *compensation inequalities and* $\mathbb{T}$ *tree patterns. In particular, we say one suite is* ***canonical*** *if all tree patterns in* $\mathbb{T}$ *contain only child axes. A canonical suite can be directly converted to a set of inequalities during the computation.*

Given any canonical suite ($\mathbb{R}$,$\mathbb{C}$,$\mathbb{T}$), the LP problem setting can be generated as follows:

$$\begin{array}{lll} \text{maximize} & \Sigma_r^{\mathcal{A}} x_r & \\ \text{subject to} & \Sigma_r^{A_R} x_r \leq 1 & for\ all\ R \in \mathbb{R} \\ & \Sigma_r^{A_P} x_r \leq 1 & for\ all\ P \in \mathbb{T} \\ & \Sigma_r^{A_C} x_r \leq 1 & for\ all\ C \in \mathbb{C} \\ & 0 \leq x_r \leq 1 & for\ all\ r \in \mathcal{A}. \end{array} \tag{5.6}$$

To generate all the canonical suites, our algorithm will recursively handle the descendant axis as follow:

1. Pick the first candidate descendant axis by preorder, else return.

2. We generate canonical suites from *conversion* for this descendant and recursively call step 1.

3. We generate canonical suites from *split* for this descendant and recursively call step 1.

The optimal size bound is to the maximum solutions from all LP inequalities from all canonical suites.

5.2.3 Two optimization rules

The above straightforward algorithm may generate $\mathcal{O}(2^n)$ canonical suites for n descendant axes in the worst-case situation. We propose some heuristics to reduce the search space for size bound computation:

- Optimization 1: Given a tree pattern T, the conversion operations in T after the split are avoidable without affecting the final result of worst-case bound.

- Optimization 2: Given a leaf descendant axis α between node x and y, assume that A denotes all labels for the root-to-y path, we define an inequality $\Sigma_r^A x_r \leq 1$. If this inequality cannot change the maximum solution for the current LP problem, then the split operation in the leaf descendant axis can be safely canceled.

This optimization may reduce the search space of the computation for the size bound. However, it does not change the exponential complexity nature of computing the size bound.

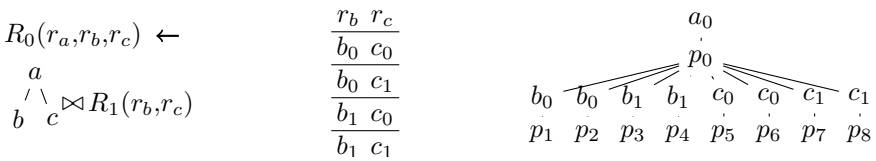
5.3 Worst-case optimal join

The first algorithm to have a running time matching these worst-case size bounds is the NPRR algorithm [67]. An important property in NPRR is to estimate the intermediate join size and avoid producing a case which is larger than the worst-case bound. In fact, for any join query, its execution time can be upper bounded by the AGM [5]. Interestingly, *LeapFrog* [78] and *Joen* [21] completely abandon the "query plan" and develop one attribute at a time to join with multiple relations. We present these results informally and refer the readers to Ngo et al. [68] for a complete survey. In this part, we discuss the challenges in designing a worst-case optimal algorithm for CMCQs over relational and tree data.

5.3.1 Tree and relational data representation

To answer a tree pattern query, a positional presentaiton of occurrences of tree elements and string values in the tree database are widely used, which extends the classic inverted index data structure in information retrieval. There existed two common ways to encode an instance tree, i.e, Dewey encoding [62] and containment encoding [12]. These decoding ways are necessary as they allow us to join tree patterns partially, thus avoiding the undesired intermediate result.

We represent the position of a string occurrence in the tree database as a Dewey code between tree nodes whose positions are recorded in the following fashion: (i) the root is labeled by a empty string ϵ; (ii) for a non-root element u, label(u)=label(s).x, where u is the x-th child of s. By this encoding, u is an ancestor of element s if and only if label(u) is a prefix of label(s). And u is a parent of element s, if and only if u is an ancestor of element s and u is one less level than s. After encoding, each attribute j in a query node can be presented



(a) Tree query Q (b) Table D_{R_1} (c) Tree D_T

Figure 5.3: A (a) CMCQ and its (b) table instance and (c) tree instance.

as a node table in the form of $t_j(r_j, p_j)$, where r_j and p_j are the label value and position value, respectively.

Later, the position value matching is obtained by the prefix matching ($\subsetneq$). When given a node table $C_{j_0}(r_{j_0}, p_{j_0})$ to be joined, we say the position value p_{j_0} is matched if we can find its parent (or ancestor) p_t such that $p_t \in t$ and $p_t \subsetneq p_{j_0}$. For example, when adopting Dewey ID for matching child and descendant relations, 1.1.1.1 is the child of 1.1.1, and the descendant of 1.1. They are matching the prefix of 1.1.1 and 1.1, respectively. Check an example from an encoded tree instance in Figure 5.3c. The position value can be added in $\mathcal{O}(N)$ by one scan of the original tree. Note that the following algorithm is not limited to the Dewey code representation. Any representation scheme which captures the structure of trees, such as region encoding [12] and extended Dewey [62] can all be used in our algorithm.

All the data in Relational are label data, and all relation tables and node tables will be expressed by the Trie index structure, which is commonly applied in the relational worst-case optimal algorithms (e.g., [2, 78]). The Trie structure can be accomplished using standard data structures (notably, balanced trees with $\mathcal{O}(\log n)$ look-up time or nested hashed tables with $\mathcal{O}(1)$ look-up time).

5.3.2 Challenges

In our context, tree data and tree pattern matching do make the situation more complex. Firstly, directly materializing tree pattern matching may yield asymptotically more intermediate results. While ignoring the pattern may loosen some bound constraints. Secondly, since tree data is represented by both label and position values, position value joining may require more computation cost for pattern matching while we don't need position values in our final result. We show an example in the following to illustrate the challenges.

Example 5.4 *Recall that a triangle relational join query* $Q = R_1(r_a, r_b) \bowtie R_2(r_a, r_c) \bowtie R_3(r_b, r_c)$ *has size bound* $\mathcal{O}(N^{1.5})$*. Figure 5.3 depicts an example of a CMCQ* Q *with the table* $R_1(r_b, r_c)$ *and twig query 'a[b]/c' to return result* $R_0(r_a, r_b, r_c)$*, which also has size bound* $\mathcal{O}(N^{1.5})$ *since PC path* a/b *and* a/c *are equivalent to constraint* $x_a + x_b \leq 1$ *and* $x_a + x_c \leq 1$*, respectively. Figure 5.3b and Figure 5.3c show the instance table* D_{R_1} *and encoded tree* D_T*. The number of label value result* $R_0(r_a, r_b, r_c)$ *is only* 4 *rows which is* $\mathcal{O}(N)$ *while the result of only the tree pattern is* 16 *rows which is* $\mathcal{O}(N^2)$*, where* N *is the table size or node size for each attribute. The final result with position values is also* $\mathcal{O}(N^2)$*. Here, the* $\mathcal{O}(N^2)$ *is from the matching result of position values of attribute* t_b *and* t_c*.*

EmptyHeaded [2] applied the existing worst-case optimal algorithms to process the graph edge pattern matching. We may also attempt to solve relation-tree joins by representing the trees as relations with the node-position and the node-label tables, and then reformulating the cross-model conjunctive query as a relational conjunctive query. However, as Example 5.4 illustrated, such a method can not guarantee the worst-case optimality as extra computation is required for position value matching in a tree.

5.3.3 CMJoin Algorithm

In this subsection, we discuss the algorithm to process both the relational and tree data. As the position values are excluded in the result set while being required for the tree pattern matching, our algorithm carefully deals with it during the join. We propose an efficient cross-model join algorithm called *CMJoin* (cross-model join). With some conditions for some queries, it guarantees the runtime optimality.

We first discover the join result size with all node position values, only branch node position value, and without position value.

Lemma 5.1 *Given relational tables* $\mathcal{R}$ *and pattern queries* $\mathcal{T}$*, let* S_r*,* S_p*, and* S'_p *be the sets of all relation attributes, all position attributes, and only branch node position attributes, respectively. It holds that*

$$\rho_1(S_r \cup S_p) \geq \rho_2(S_r \cup S'_p) \geq \rho_3(S_r) \tag{5.7}$$

Proof. $Q(S_r)$ is the projection result from $Q(S_r \cup S'_p)$ by removing all position values, and $Q(S_r \cup S'_p)$ is the projection result from $Q(S_r \cup S_p)$ by removing non-branch position values. Therefore, the result size holds $\rho_1(S_r \cup S_p) \geq \rho_2(S_r \cup S'_p) \geq \rho_3(S_r)$. □

Example 5.5 *Continue the CMCQ Q in Figure 5.3a, which is* $Q{=}R_1(r_b, r_c) \bowtie a[b]/c$. *Node a, b, c in the tree pattern can be presented as node tables* (r_a, p_a), (r_b, p_b), *and* (r_c, p_c), *respectively. So we have* $Q(S_r \cup S_p) = R(r_a, r_b, r_c, p_a, p_b, p_c)$, $Q(S_r \cup S'_p) = R(r_a, r_b, r_c, p_a)$, *and* $Q(S_r) = R(r_a, r_b, r_c)$. *Then by LP constraint bound by the relations and PC-paths, we can achieve* $\mathcal{O}(N^2)$, $\mathcal{O}(N^{\frac{3}{2}})$, *and* $\mathcal{O}(N^{\frac{3}{2}})$ *for the size bound* $\rho_1(S_r \cup S_p)$, $\rho_2(S_r \cup S'_p)$, *and* $\rho_3(S_r)$, *respectively.*

Specifically for *CMJoin*, we elaborate more in the following:

- If $\rho_1(S_r \cup S_p){=}\rho_3(S_r)$, then *CMJoin* executes a generic relational worst-case optimal join algorithm [2, 21].
- Else, *CMJoin* computes the path result of the tree pattern first. We project out all position values of non-branch nodes for the query tree pattern. Then, we keep the position values of only branch nodes so that we still can match the whole part of the tree pattern.

Theorem 5.2 *Given relations* $\mathcal{R}$ *and pattern queries* $\mathcal{T}$, *If (1)* $\rho_1(S_r \cup S_p) \leq \rho_3(S_r)$; *or (2) (i)* $\rho_1(S_r \cup S'_p) \leq \rho_3(S_r)$ *and (ii) for each path P in* $\mathcal{T}$, *let* S''_r *and* S''_p *be the set of label and position attributes for P,* $\rho_4(S''_r \cup S''_p) \leq \rho_3(S_r)$; *Then, CMJoin is worst case optimal to* $\rho_3(S_r)$.

Proof. (1) As $\rho_3(S_r)$ is the projection of $\rho_1(S_r \cup S_p)$, we first compute $Q(S_r \cup S_p)$, then project out all the position value in linear of result size. Since $\rho_1(S_r \cup S_p) \leq \rho_3(S_r)$, so the result size is less than $\mathcal{O}(N^{\rho_3(S_r)})$.

(2) $\rho_1(S''_r \cup S''_p) \leq \rho_3(S_r)$ means that each path result with label and position values are less than $\rho_3(S_r)$. We first compute the path result and then project out all the non-branch position values under $\rho_3(S_r)$. And $\rho_2(r, p') \leq \rho_3(r)$ means that with all branch position values in the join result, its worst case result size is still under $\rho_3(S_r)$. Then by considering those position values as relational attribute values and by a generic relation join, *CMJoin* is worst-case optimal to $\rho_3(S_r)$. □

Example 5.6 *Continue the CMCQ Q in Figure 5.3a, which is* $Q{=}R_1(r_b, r_c) \bowtie a[b]/c$. *As* $\rho_1(S_r \cup S_p) > \rho_3(r)$, *directly computing all label and position values generate* $\mathcal{O}(N^2)$ *result. Instead, we can compute the path result of* a/b *and* a/c, *which are* (r_a, p_a, r_b, p_b) *and* (r_a, p_a, r_c, p_c) *and in* $\mathcal{O}(N)$. *Then we obtain only the branch node result* (r_a, p_a, r_b) *and* (r_a, p_a, r_c) *by projecting out the non-branch node position values. Then joining the processed result with relation* R_1 *by a generic worst-case optimal algorithm, we can guarantee the intermediate and final result are* $\mathcal{O}(N^{\frac{3}{2}})$.

5.4 Evaluation

In this section, we experimentally evaluate the performance of the proposed algorithm, *CMJoin* with four real-life and benchmark data sets, against state-of-the-art systems and algorithms concerning efficiency, scalability, and intermediate cost.

5.4.1 Evaluation setup

Datasets and query design Table 5.1 shows the statistics of datasets and designed CMCQs. These diverse datasets are different from the tree structure, data skewness, data size, and data model varieties. Accordingly, we design 24 CMCQs for the evaluation. Picking diverse sources and designing proper queries evaluate the efficiency, scalability, and cost performance of the *CMJoin* in various real-world scenarios.

Comparison systems and algorithms *CMJoin* is compared with two types of state-of-the-art cross-model solutions:

- The first solution is to use one query to retrieve from the system without changing the nature of models [66,90]. We implemented queries in two mainstream systems that support cross-model join, one open-source database, i.e., *Postgres* (*PG*), and one commercial database (*CDB* anonymously due to lacking licenses). They both enable the default query optimizer.
- The second is to encode and retrieve tree nodes in a relational engine [2, 11, 69, 93]. We implemented two algorithms, i.e., structure join (*SJ*) (pattern matching first) and value join (*VJ*) (label value matching first). In addition, we also compared to a worst-case optimal relational engine called EmptyHeaded (*EH*) [2].

Experiment Setting We implemented our solutions in Python 3 and conducted all experiments on a 64-bit Windows machine with a 4-core Intel i7-4790 CPU at 3.6 GHz, 16 GB RAM, and 500 GB HDD. We measure the computation time of joining as the main metric, excluding the time used for compilation, data loading, index presorting, and representation/index creation for all systems and algorithms. We employed Dewey encoding [62] in all experiments. The joining order of attributes is greedily chosen based on the frequency of attributes. We measure the intermediate cost metric by accumulating all intermediate and final joining results. For *PG*, we accumulate all sub-query intermediate results. We

Table 5.1: Dataset statistics and designed queries (m=10^6, k=10^3).

Dataset	Statistics	Query	Relational table	XML or JSON path query	LP	#Result
D1:TreeBank [86] (Linguistic data)	Zipfian Tables: $1m$ rows XML: $2.4m$ nodes	Q1	R1(NP,VP)	S[NP]/VP//PP[IN]//NNP	N^3	$7.6k$
		Q2	R1(NP,VP) R2(NP,PP)		N^3	$4.6k$
		Q3	R1(NP,VP) R3(NP,NNP)		N^3	¡$0.1k$
		Q4	R1(NP,VP)	S[NP]/VP//PP[IN]//NNP S/VP/PP/IN	N^3	$1.4k$
		Q5	R1(NP,VP) R2(NP,PP)		N^2	$0.8k$
		Q6	R1(NP,VP) R3(NP,NNP)		N^3	¡$0.1k$
D2:Xmark [72] (Auction data)	Normal Tables: $1m$ rows XML: $1.6m$ nodes	Q7	R1(incategory,quantity,email) R2(item,incategory,email) R3(item,quantity,email)	T7=Item[incategory]/quantity	N^2	$91k$
		Q8		T8=Item[incategory][localtion][quantity]//email	N^3	$0.4k$
		Q9		T9=Item[location]//email	N^3	$2.4m$
		Q10		T7, T8	N^3	$0.7k$
		Q11		T7, T9	N^3	$0.7k$
		Q12		T7, T8, T9	N^3	$0.7k$
D3:UniBench [90] (E-commerce)	Uniform Tables: $1m$ rows JSON: $2m$-$4m$ nodes	Q13	R1(asin,productID,orderID) R2(personID,lastname) R3(productID,product_info)	$.[orderID,personID]	N^3	$37.0m$
		Q14		$.[orderID,personID,orderline[productID]]	N^3	¡$0.1k$
		Q15		$.[personID,orderline[productID, asin]]	N^3	$0.1k$
D3:UniBench [90] (E-commerce)	Uniform Tables: $1m$ rows JSON: $4m$ nodes XML: $1.4m$ nodes	Q16	R1(asin,orderID) R2(personID,lastname) $.[orderID,personID,orderline[asin]]	OrderLine[asin]/price	N^3	$1.1k$
		Q17		T17=Invoice[orderID]/orderline[asin]/price	N^3	¡$0.1k$
		Q18		T18=Invoice[orderID]//asin	N^3	¡$0.1k$
		Q19		orderline/asin, orderline/price	N^3	$1.1k$
		Q20		Invoice(I)/orderID, I/orderline(O)/asin, I/O/price	N^3	¡$0.1k$
		Q21		T17, T18	N^3	¡$0.1k$
D4:MIMIC-III [51] (Clinical data)	Uniform Tables:0.5-$10m$ rows JSON: $10m$ nodes	Q22	R1(RowID,ICUstayID,ItemID,CGID), R2(RowID,SubjectID,ICUstayID,ItemID)	T22=$.[RowID,SubjectID,HADMID]	N	¡$0.1k$
		Q23	R1,R3(SubjectID,ItemID)	T22	$N^{\frac{3}{2}}$	¡$0.1k$
		Q24	R1,R2,R4(RowID,SubjectID,HADMID)	T22, T23=$.[RowID,ICUstayID,ItemID,CGID], T24=$.[RowID,SubjectID,HADMID, ICUstayID,ItemID,CGID]	N	¡$0.1k$

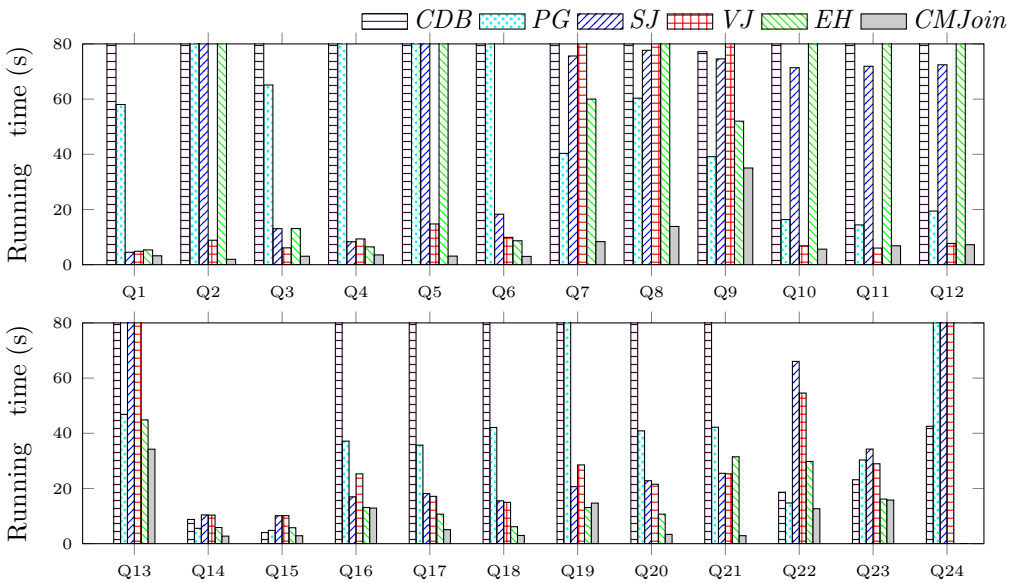


Figure 5.4: Efficiency: runtime performance for all queries by CDB, PG, SJ, VJ and $XJoin$. The performance time of more than 80s is cut for better presentation.

repeat five experiments excluding the lowest and the highest measure, and obtain the average results. Between each measurement of queries, we wipe the caches and re-load the data to avoid intermediate results.

5.4.2 Evaluation of CMJoin

We show the evaluation results of *CMJoin* against state-of-the-art systems (*CDB* and *PG*)) and algorithms (*SJ*, *VJ*, and *EH*) via rich datasets and queries in terms of the efficiency, scalability, and intermediate cost.

Efficiency Figure 5.4 shows the efficiency evaluation. In general, *CMJoin* averagely outperforms other solutions 3.33-13.43x (details in Table 5.2). These numbers are conservative as we omit the "out of memory" (OOM) and "time out" (TO) results in the calculation. Algorithm solutions *SJ*, *VJ*, and *EH* outperform *CDB* and *PG* in the majority of the cases as they decompose the tree data into relation-like and smaller-modular formats, speedup the node retrieving and tree pattern matching. *CDB* performs worst in most of the cases, even comparing to *PG* as *PG* has a better heuristic optimizer to optimize the joining processing.

Specifically in $Q1$-$Q6$, *CMJoin*, *SJ*, *VJ*, and *EH* outperform *CDB* and *PG*. The reason is that the original tree is deeply recursive in the TreeBank dataset [86], and designed tree pattern queries are complex. Therefore, it is costly to retrieve results directly from the original tree by *CDB* and *PG* by original tree structure. Instead, *CMJoin*, *SJ*, and *VJ* by using encoded structural information are convenient to retrieve nodes and match tree patterns in such cases. In $Q2$ and $Q5$, *EH* perform worse. The reason is that it seeks for better instance bound by joining partial tables and sub-twig (partial tree pattern) first, then aggregating the result. However, these separated partial joins generate more intermediate results in this dataset. In $Q1$ and $Q4$, with only one table, *SJ* and *VJ* perform similarly as table joining does not occur in both cases. However, in $Q2$-$Q3$ and $Q5$-$Q6$, *SJ* performs worse as joining two tables first leads to massive intermediate results in this dataset.

By contrast, *SJ* outperforms *VJ* in $Q7$-$Q9$. This time, the Xmark dataset [72], the data tree is flat and with less matching results in tree pattern queries. In tables, the data are less skewed. Therefore, *SJ* operates table joins and tree pattern matching separately, generating less results. Instead, *VJ* considers tree pattern matching later, generating a massive intermediate result (more details of $Q7$ in Figures 5.5 and 5.6) when joining label values between two models with the non-uniform data. *PG*, implementing in a similar way as *SJ*, performs satisfactorily as well. The above comparisons show that the compared solutions, which can achieve superiority only in some cases, can not adapt well to dataset dynamics.

Meanwhile, we continue with the same dataset with different queries in $Q10$-$Q12$, which have more complex tree pattern nodes involved, *VJ* filters more values and produces less intermediate results, thus outperforming *SJ* ($\sim$10x) and *PG* ($\sim$2x). While for *EH*, it also generates massive intermediate results for queries (e.g., $Q10$-$Q12$) with more connections (joins) in attributes. The comparison between $Q7$-$Q9$ and $Q10$-$Q12$ indicates the compared solutions can not adapt well to query dynamics.

In $Q13$-$Q15$ and $Q22$-$Q24$, *CDB* and *PG* perform relatively better, as this time, it involves only JSON and relational data. *CDB* and *PG* perform well in JSON retrieving as these JSON documents have a simpler structure than XML. All solutions perform reasonably well when the result size is small in $Q14$-$Q15$ while *CDB*, *SJ*, and *VJ* still suffer from larger result size in $Q13$. With only JSON data, *SJ* and *VJ* perform similarly, as they both treat a simple JSON tree as one relation. In contrast in $Q16$-$Q21$, it involves XML, JSON and relational data of the UniBench dataset [90]. *CMJoin*, *SJ*, *VJ*, and *EH* perform better

Table 5.2: Intermediate result size (10^6) and running time (S) for queries. "/" and "-" indicate "time out" ($\geq$ 10 mins) and "out of memory", respectively.

	Intermediate result size(10^6)					Running time (S)					
Query	**PG**	**SJ**	**VJ**	**EH**	***CMJoin***	**CDB**	**PG**	**SJ**	**VJ**	**EH**	***CMJoin***
Q1	7.87x	2.60x	2.00x	1.68x	0.15	153.5x	18.02x	1.39x	1.51x	1.66x	3.22
Q2	/	-	3.75x	4.83x	0.08	/	/	-	4.52x	129x	1.96
Q3	86.0x	62.6x	3.63x	4.61x	0.08	/	21.3x	4.27x	1.99x	4.28x	3.06
Q4	/	1.96x	1.75x	1.64x	0.24	/	/	2.34x	2.63x	1.82x	3.55
Q5	/	-	1.86x	1.77x	0.22	/	/	-	4.75x	39.8x	3.11
Q6	/	2.24x	2.00x	1.85x	0.21	/	/	6.10x	3.30x	2.89x	3.00
Q7	133x	106x	-	35.1x	0.29	15.4x	4.82x	9.05x	-	7.18x	8.36
Q8	350x	279.8x	-	/	0.11	/	4.36x	5.61x	-	/	13.8
Q9	8.87x	8.34x	-	2.01x	4.62	2.20x	1.12x	2.13x	-	1.48x	35.0
Q10	110x	440x	4.86x	/	0.07	/	2.91x	12.7x	1.22x	/	5.62
Q11	110x	440x	4.86x	/	0.07	/	2.11x	10.5x	0.88x	/	6.84
Q12	110x	440x	4.86x	/	0.07	/	2.68x	9.99x	1.06x	/	7.25
Q13	1.04x	1.22x	1.22x	1.07x	43.2	6.87x	1.37x	4.81x	4.79x	1.31x	34.2
Q14	19.7x	2.56x	2.56x	3.90x	0.39	3.22x	2.04x	3.82x	3.79x	2.14x	2.73
Q15	14.2x	1.85x	1.85x	17.0x	0.54	1.40x	1.68x	3.53x	3.54x	2.01x	2.87
Q16	1.24x	1.24x	6.81x	2.15x	0.37	/	2.88x	1.32x	1.96x	1.02x	12.9
Q17	1.59x	7.84x	2.28x	1.31x	0.32	/	7.03x	3.58x	3.38x	2.10x	5.08
Q18	1.59x	7.13x	1.59x	1.64x	0.32	/	14.1x	5.21x	5.02x	2.06x	2.98
Q19	/	5.47x	6.62x	1.77x	0.45	/	/	1.41x	1.94x	0.89x	14.7
Q20	7.80x	25.1x	7.30x	4.19x	0.10	/	12.1x	6.77x	6.39x	3.17x	3.37
Q21	12.0x	36.1x	18.4x	14.7x	0.10	/	14.6x	8.82x	8.75x	10.9x	2.89
Q22	1.00x	18.5x	18.5x	0.96x	0.57	1.47x	1.16x	5.22x	4.31x	2.35x	12.7
Q23	18.5x	18.5x	18.5x	1.61x	0.57	1.46x	1.92x	2.17x	1.83x	1.02x	15.8
Q24	14.3x	3.02x	4.02x	0.96x	0.57	>4kx	>9kx	>11kx	>12kx	0.18x	0.01
AVG	5.46x	5.90x	1.92x	1.90x	2.24	13.43x	4.37x	5.34x	3.33x	3.46x	8.54

than *CDB* and *PG*. This is again because employing the encoding technique in trees accelerates retrieving nodes and matching tree patterns. Also, *CMJoin*, *SJ*, *VJ*, and *EH* can treat all the data models together instead of achieving results separately from each model by queries in *CDB* and *PG*.

Though the compared systems and algorithms possess their advantages of processing and matching data, they straightforwardly join without bounding intermediate results, thus achieving sub-optimal performance during joining. *CMJoin* is the clear winner against other solutions, as *CMJoin* can wisely join between models and between data to avoid unnecessary quadratic intermediate results.

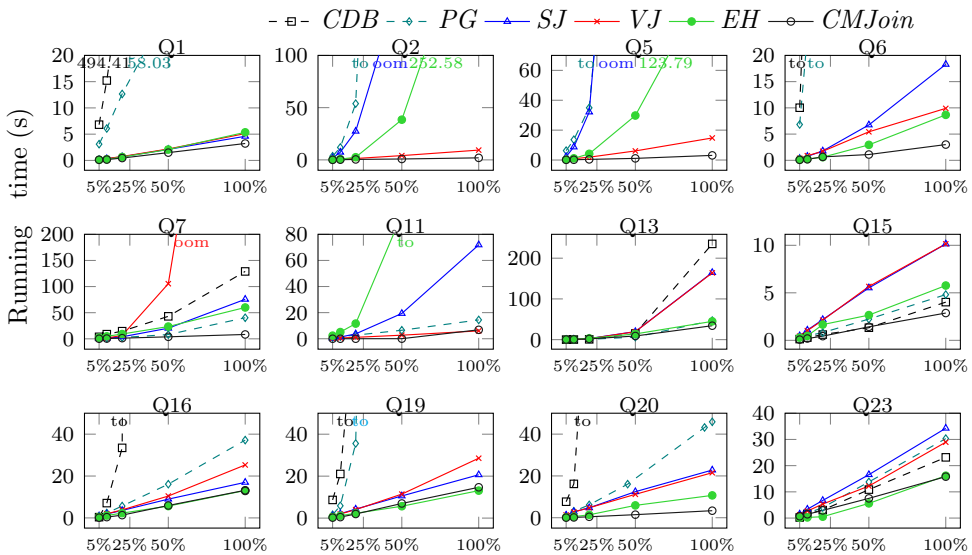


Figure 5.5: Scalability: runtime performance by *CDB*, *PG*, *SJ*, *VJ* and *CMJoin*. The x-axis is the percentage of data size. "oom" and "to" stand for "out of memory" error and "time out" ($\geq$ 10 mins), respectively.

Scalability Figure 5.5 shows the scalability evaluation for some queries. Some results are similar and less interesting, so omitted. In most queries, as the data size increases, *CMJoin* performs flatter scaling, since *CMJoin* is designed to control the unnecessary intermediate output.

As discussed, *CMJoin*, *SJ*, *VJ*, and *EH* outperform *CDB* and *PG* in most of the queries, as the encoding method of the algorithms speeds up the tree pattern matching especially when the documents or queries are more complex. However, *CDB* and *PG* scales better when involving simpler documents (e.g., in $Q15$ and $Q23$) or simpler queries (e.g., in $Q7$). Compared to processing XML tree pattern queries, the systems *CDB* and *PG* process JSON data more efficiently.

Interestingly in $Q2$, *SJ* and *PG* joins two relational tables separately from twig matching, generating quadratic intermediate results, thus leading to the OOM and TO, respectively. While in $Q7$, *VJ* joins tables with label values without considering tree pattern matching and outputs an undesired non-linear increase of intermediate results, thus leading to OOM in larger data size. Likewise evaluating *EH* between $Q2$ and $Q7$, it can not adapt well with different datasets. Performing differently in diverse datasets between SJ/PG and VJ/EH indicates that they

cannot smartly adapt to dataset dynamics. While increasing tree pattern queries in $Q11$ compared to $Q7$, *VJ* filters more results and thus decreases the join cost and time in $Q11$. The comparison between SJ/EH and *VJ* having dramatically different performances in the same dataset with different queries indicates that they cannot smartly adapt to query dynamics.

In $Q11$, both *CMJoin* and *VJ* perform efficiently as they can filter out most of the values at the beginning. In this case, *CMJoin* runs slightly slower than *VJ*, which is reasonable as *CMJoin* maintains a tree structure rather than only tuple results in *VJ*. Overall, *CMJoin* judiciously joins between models and controls unwanted massive intermediate results, thus performing efficiently and stably in not only dynamic datasets and queries but also data scaling.

Cost analysis Table 5.2 also presents the intermediate result sizes, showing that *CMJoin* outputs 5.46x, 5.90x, 1.92x, and 1.90x on average less intermediate results than *PG*, *SJ*, *VJ*, and *EH*, respectively. Results of *CDB* in this part are less interesting and omitted as half of the queries end up TO. Figure 5.6 depicts more detailed intermediate results for each joining step. In general, *CMJoin* generates less intermediate results due to its designed algorithm process, worst-case optimality, as well as joining order selections. In contrast, *PG*, *SJ*, and *VJ* can easily generate massive (often quadratic) intermediate results during the joining process in different datasets or queries. This is because they have no technique to avoid undesired massive intermediate results.

PG and *SJ* suffer when the tree pattern matching becomes complex in datasets (e.g., $Q3$ and $Q10$), while *VJ* suffers in the opposite case of simpler twig pattern matching (e.g., $Q7$ and $Q16$). More specifically in $Q3$, *PG* and *SJ* output significantly more intermediate results by joining of two relational tables while *VJ* controls intermediate results by the values of common attributes or tags between two models. Meanwhile in $Q7$, $Q9$, and $Q16$, *VJ* does not consider structural matching at first, yielding unnecessary quadratic intermediate results. The above two-side examples indicate that the compared solutions considering only one model at a time or joining values first without twig matching produce an undesired significant intermediate result.

EH suffers when the queries/attributes are more connected (joined), leading to more intermediate results during joining procedures. As *EH* seeks better instance bound, it follows the query plan based on the GHD decomposition [2]. Our proposed algorithm, *CMJoin*, by wisely joining between models, avoids the unnecessary massive intermediate output from un-joined attributes.

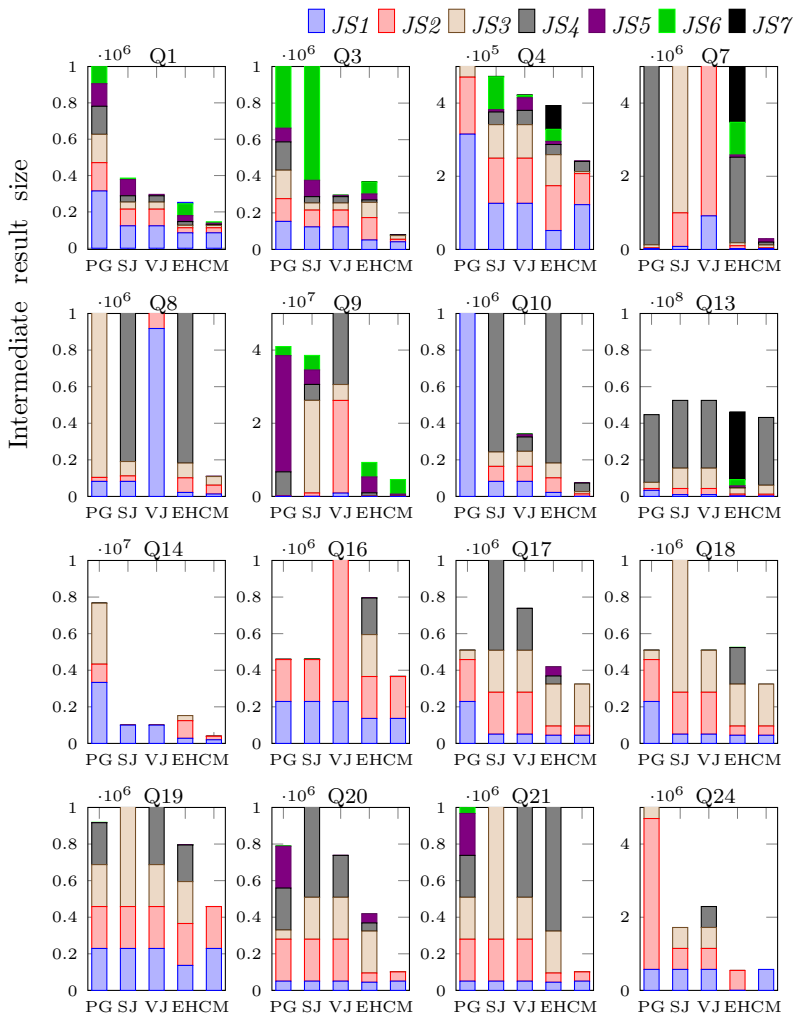


Figure 5.6: Cost: intermediate result size during joining. CM: *CMJoin*. JS: Joining step.

Summary We summarize evaluations of *CMJoin* as follows:

- Extensive experiments on diverse datasets and queries show that averagely, *CMJoin* against the compared solutions achieves up to 13.43x faster runtime performance and produces up to 5.46x less intermediate results.
- With skew data, *CMJoin* avoids undesired huge intermediate results by wisely joining data between models. With uniform data, *CMJoin* filters out more values by joining one attribute at a time between all models.
- With more tables, tree patterns, or common attributes involved, *CMJoin* seems to perform more efficiently and scalably.

5.5 Chapter summary

This chapter introduced the topic of the worst-case optimal join of Papers V and VI, in which we initialize the problem of worst-case optimality for relational and tree data to answer RQ3 and RQ4. We first defined the problem of size bound for both relational and tree data. We also proposed to compute the size bound for them. Then we proposed the *CMJoin* algorithm to efficiently process the relational and tree data joins, compared to state-of-the-art solutions and systems.

Chapter 6

Conclusions and future work

In this chapter, we conclude this thesis by revisiting Chapters 3 to 5. Furthermore, we present some interesting research directions that might be worth investigating further.

6.1 Conclusions

As the increasing of heterogeneous data created by all kinds of applications, taking advantages of these raw data and adding values to them is in the interests of all researchers, engineers, and customers. However, it is extremely tough to cost-effectively store, collect, process, query, and analyze large-scale data.

In this thesis, we advanced approaches to optimize the performance of big data systems and queries. Specifically, we discussed two main topics. The first topic is performance tuning for big data systems. For performance tuning, we first leveraged cost-model and triangulation to model the performance, then made a cost-effective prediction based on these models. The second topic is the worst-case optimal algorithm for join queries. We researched the size bound and worst-case optimal join algorithm for the case with both relational and tree data in contrast with only relations.

In Chapter 3, we introduced a detailed cost model for Spark workloads, which leverages Monte Carlo simulation to achieve cost-effective training. Specifically, it utilizes a little part of resources and data to make a reliable prediction for larger clusters and datasets, even with data skewness and runtime deviations. Also, this work considers network and disk bounds so that it performs better with I/O-bounded workloads.

In Chapter 4, we proposed *d*-simplexed, which models the Spark workloads by leveraging Delaunay Triangulation. Unlike other black-box approaches, *d*-simplexed utilizes piece-wise linear regression models, which can be built faster and yield better prediction. Also, *d*-simplexed is built with an adaptive sampling technique which collects only a few training points to achieve accurate prediction.

In Chapter 5, we first embarked on the study of the cross-model conjunctive query (CMCQ) with relational and tree data, and formally defined the problem of CMCQ processing. We revealed that the computation of the worst-case size bound of a CMCQ is $\mathcal{NP}$-hard w.r.t query expression complexity. We developed a worst-case optimal join algorithm called *CMJoin* to match the size bound of a CMCQ under some circumstances.

6.2 Future work

As the first direction, the workloads may be a variant and cannot be exactly the same as we trained before. It would be interesting to investigate a way to express the characteristics of the workloads. For instance, we can name a CPU-intensive workload when the ratio of the computation time is more than 50% of the overall completion time. A workload can be described as I/O intensive when the ratio of the time of reading/writing from/in disk is more than 50% of the overall completion time. The ratio of computing time, read/write time, or shuffle time are reliable indicators for users to set up or configure cluster resources beforehand. Also, such characteristics may help the scheduler to assign bottleneck resources accordingly. Our framework is tested and shown effectiveness. However, the prediction accuracy may not be sufficient in all workloads and scenarios. It would be interesting to explore more complex workloads such as deep learning algorithms. Also, modeling performance in the competition of non-configurable resources (e.g., disk and network) with multiple parallel workloads would be challenging but promising.

Another research direction would be extending the approaches for finding better joining order for join algorithms. Though any order does not affect the worst-case optimality for join algorithms, a better plan [28, 34] may lead to a better bound for some instances [2] by combining the worst-case optimal algorithm and non-cyclic join optimal algorithm (i.e. Yannakakis [87]). Also, a worst-case optimal join algorithm does not guarantee the best performance for some cases. A combined method may be able to find a better join plan based on some statistics. We leave this as the future work to continue optimizing CMCQs.

References

[1] Apache Spark REST API. `https://spark.apache.org/docs/latest/monitoring.html`.

[2] Christopher R. Aberger, Susan Tu, Kunle Olukotun, and Christopher Ré. Emptyheaded: A relational engine for graph processing. In *International Conference on Management of Data (SIGMOD)*, pages 431–446. ACM, 2016.

[3] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *International Conference on Management of Data (SIGMOD)*, pages 1009–1024. ACM, 2017.

[4] Sterling J. Anderson, Sisir Karumanchi, and Karl Iagnemma. Constraint-based planning and control for safe, semi-autonomous operation of vehicles. In *Intelligent Vehicles Symposium*, pages 383–388. IEEE, 2012.

[5] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. In *49th Annual IEEE Symposium on Foundations of Computer Science*, pages 739–748. IEEE Computer Society, 2008.

[6] Ahsan Javed Awan, Mats Brorsson, Vladimir Vlassov, and Eduard Ayguadé. Architectural impact on performance of in-memory data analytics: Apache spark case study. *CoRR*, abs/1604.08484, 2016.

[7] Liang Bao, Xin Liu, and Weizhao Chen. Learning-based automatic parameter tuning for big data analytics frameworks. In *IEEE International Conference on Big Data (Big Data)*, pages 181–190. IEEE, 2018.

[8] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[9] Muhammad Bilal and Marco Canini. Towards Automatic Parameter Tuning of Stream Processing Systems. In *Proceeding of the 8th ACM Symposium on Cloud Computing (SoCC)*, pages 189–200. ACM, 2017.

[10] Kurt Binder. Monte Carlo simulations in statistical physics. In *Encyclopedia of Complexity and Systems Science*, pages 5667–5677. Springer, 2009.

[11] Zakaria Bousalem and Ilias Cherti. Xmap: A novel approach to store and retrieve XML document in relational databases. *Journal of Software (JSW)*, 10(12):1389–1401, 2015.

[12] Nicolas Bruno, Nick Koudas, and Divesh Srivastava. Holistic twig joins: optimal XML pattern matching. In *International Conference on Management of Data (SIGMOD)*, pages 310–321. ACM, 2002.

[13] Victor Bryant. Linear programming duality; an introduction to oriented matroids. *The Mathematical Gazette*, 77(480):387–387, 1993.

[14] Hao Chen and Jonathan Bishop. Delaunay triangulation for curved surfaces. *Meshing Roundtable*, pages 115–127, 1997.

[15] Keke Chen, James Powers, Shumin Guo, and Fengguang Tian. CRESP: towards optimal resource provisioning for mapreduce computing in public clouds. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1403–1412, 2014.

[16] Yuxing Chen. Worst case optimal joins on relational and XML data. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1833–1835, 2018.

[17] Yuxing Chen, Peter Goetsch, Mohammad A Hoque, Jiaheng Lu, and Sasu Tarkoma. d-simplexed: Adaptive delaunay triangulation for performance modeling and prediction on big data analytics. *Preprint, IEEE Transactions on Big Data*, 2019.

[18] Yuxing Chen, Mohammad A. Hoque, Pengfei Xu, Jiaheng Lu, and Sasu Tarkoma. Cost-effective resource provision prediction and recommendation for spark workloads. *Submitted to Journal of Distributed and Parallel Databases (DPD)*, 2021.

[19] Yuxing Chen and Jiaheng Lu. Worst-case optimal algorithms for cross-model conjunctive queries. *Submitted to Conference on Database Systems for Advanced Applications (DASFAA)*, 2021.

[20] Yuxing Chen, Jiaheng Lu, Chen Chen, Mohammad Hoque, and Sasu Tarkoma. Cost-effective resource provisioning for spark workloads. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 2477–2480, 2019.

[21] Radu Ciucanu and Dan Olteanu. Worst-case optimal join at a time. Technical report, Technical report, Oxford, 2015.

[22] Miyuru Dayarathna and Srinath Perera. Recent Advancements in Event Processing. *ACM Computing Surveys*, 51(2):33, 2018.

[23] Jesús A De Loera, Jörg Rambau, and Francisco Santos. *Triangulations Structures for algorithms and applications*. Springer, 2010.

[24] B. Delaunay. Sur la sphère vide. a la mémoire de georges voronoï. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na*, pages 793–800, 1934.

[25] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. Tuning Database Configuration Parameters with iTuned. *The Proceedings of the VLDB Endowment (PVLDB)*, 2(1):1246–1257, 2009.

[26] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. Tuning database configuration parameters with ituned. *The Proceedings of the VLDB Endowment (PVLDB)*, 2(1):1246–1257, 2009.

[27] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magdalena Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stanley B. Zdonik. The bigdawg polystore system. *ACM SIGMOD Record*, 44(2):11–16, 2015.

[28] Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and fractional hypertree decompositions: Hard and easy cases. In *the Symposium on Principles of Database Systems (PODS)*, pages 17–32. ACM, 2018.

[29] Mikhail Genkin, Frank Dehne, Maria Pospelova, Yabing Chen, and Pablo Navarro. Automatic, on-line tuning of YARN container memory and CPU

parameters. In *International Conference on High Performance Computing and Communications; International Conference on Smart City; International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 317–324. IEEE Computer Society, 2016.

[30] Paul-Louis George and Houman Borouchaki. Delaunay triangulation and meshing. *Hermes, Paris, France*, 1998.

[31] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2011.

[32] Tomasz Gogacz and Szymon Toruńczyk. Entropy bounds for conjunctive queries with functional dependencies. *arXiv preprint arXiv:1512.01808*, 2015.

[33] Tomasz Gogacz and Szymon Toru'nczyk. Entropy bounds for conjunctive queries with functional dependencies. In *International Conference on Database Theory (ICDT)*, pages 15:1–15:17, 2017.

[34] Georg Gottlob, Martin Grohe, Nysret Musliu, Marko Samer, and Francesco Scarcello. Hypertree decompositions: Structure, algorithms, and applications. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 1–15. Springer, 2005.

[35] Georg Gottlob, Christoph Koch, and Klaus U. Schulz. Conjunctive queries over trees. *Journal of ACM*, 53(2):238–272, 2006.

[36] Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. Size and treewidth bounds for conjunctive queries. *Journal of ACM*, 59(3):16:1–16:35, 2012.

[37] Anastasios Gounaris, Georgia Kougka, Ruben Tous, Carlos Tripiana Montes, and Jordi Torres. Dynamic Configuration of Partitioning in Spark Applications. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 28(7):1891–1904, 2017.

[38] Anastasios Gounaris, Georgia Kougka, Rubén Tous, Carlos Tripiana Montes, and Jordi Torres. Dynamic configuration of partitioning in spark applications. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):1891–1904, 2017.

[39] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1), August 2014.

[40] Neil J. Gunther, Paul Puglia, and Kristofer Tomasette. Hadoop superlinear scalability. *ACM Queue*, 13(5):20, 2015.

[41] Rihan Hai, Sandra Geisler, and Christoph Quix. Constance: An intelligent data lake system. In *International Conference on Management of Data (SIGMOD)*, pages 2097–2100. ACM, 2016.

[42] Álvaro Brandón Hernández, María S Perez, Smrati Gupta, and Victor Muntés-Mulero. Using machine learning to optimize parallelism in big data applications. *Future Generation Computer Systems*, 86:1076–1092, 2018.

[43] Álvaro Brandón Hernández, María S. Pérez, Smrati Gupta, and Victor Muntés-Mulero. Using machine learning to optimize parallelism in big data applications. *Future Generation Computer Systems*, 86:1076–1092, 2018.

[44] Herodotos Herodotou, Yuxing Chen, and Jiaheng Lu. A survey on automatic parameter tuning for big data processing systems. *ACM Computing Survey*, 53(2):43:1–43:37, 2020.

[45] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. Starfish: A self-tuning system for big data analytics. In *Proceeding of the 5th Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 261–272, 2011.

[46] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. Starfish: A self-tuning system for big data analytics. In *The Conference on Innovative Data Systems Research (CIDR)*, pages 261–272, 2011.

[47] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *IEEE International Conference on Data Engineering Workshops (ICDEW)*, pages 41–51. IEEE Computer Society, 2010.

[48] Ronald L Iman. Latin hypercube sampling. *Wiley StatsRef: Statistics Reference Online*, 2008.

[49] Pooyan Jamshidi and Giuliano Casale. An Uncertainty-aware Approach to Optimal Configuration of Stream Processing Systems. In *International*

Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pages 39–48. IEEE, 2016.

[50] Zhen Jia, Chao Xue, Guancheng Chen, Jianfeng Zhan, Lixin Zhang, Yonghua Lin, and Peter Hofstee. Auto-tuning spark big data workloads on POWER8: prediction-based dynamic SMT threading. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 387–400. ACM, 2016.

[51] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. MIMIC-III, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

[52] Selvi Kadirvel and José AB Fortes. Grey-box Approach for Performance Prediction in Map-Reduce based Platforms. In *International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9. IEEE, 2012.

[53] David J Ketchen and Christopher L Shook. The application of cluster analysis in strategic management research: an analysis and critique. *Strategic management journal*, 17(6):441–458, 1996.

[54] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing join queries with functional dependencies. In *the Symposium on Principles of Database Systems (PODS)*, pages 327–342. ACM, 2016.

[55] Rahul Krishna, Chong Tang, Kevin J. Sullivan, and Baishakhi Ray. Conex: Efficient exploration of big-data system configurations for better performance. *CoRR*, abs/1910.09644, 2019.

[56] Min Li, Liangzhao Zeng, Shicong Meng, Jian Tan, Li Zhang, Ali Raza Butt, and Nicholas C. Fuller. MRONLINE: mapreduce online performance tuning. In *International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pages 165–176. ACM, 2014.

[57] Guangdeng Liao, Kushal Datta, and Theodore L Willke. Gunther: Search-based Auto-tuning of MapReduce. In *Proceeding of the European Conference on Parallel Processing (Euro-Par)*, pages 406–419. Springer, 2013.

[58] Lynn H Loomis and Hassler Whitney. An inequality related to the isoperimetric inequality. *Bulletin of the American Mathematical Society*, 55(10):961–962, 1949.

[59] Jiaheng Lu, Yuxing Chen, Herodotos Herodotou, and Shivnath Babu. Speedup your analytics: Automatic parameter tuning for databases and big data systems. *The Proceedings of the VLDB Endowment (PVLDB)*, 12(12):1970–1973, 2019.

[60] Jiaheng Lu and Irena Holubová. Multi-model data management: What's new and what's next? In *International Conference on Extending Database Technology (EDBT)*, pages 602–605. OpenProceedings.org, 2017.

[61] Jiaheng Lu and Irena Holubová. Multi-model databases: A new journey to handle the variety of data. *ACM Computing Survey*, 52(3):55:1–55:38, June 2019.

[62] Jiaheng Lu, Tok Wang Ling, Chee Yong Chan, and Ting Chen. From region encoding to extended dewey: On efficient processing of XML twig pattern matching. In *The Proceedings of the VLDB Endowment (PVLDB)*, pages 193–204. ACM, 2005.

[63] Jiaheng Lu, Zhen Hua Liu, Pengfei Xu, and Chao Zhang. UDBMS: road to unification for multi-model data management. *CoRR*, abs/1612.08050, 2016.

[64] Carla C Morris and Robert M Stark. *Finite Mathematics: Models and Applications*. John Wiley & Sons, 2015.

[65] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. Using bad learners to find good configurations. In *the European Software Engineering Conference/Foundations on Software Engineering (ESEC/FSE)*, pages 257–267. ACM, 2017.

[66] Hassana Nassiri, Mustapha Machkour, and Mohamed Hachimi. One query to retrieve XML and relational data. In *International Conference on Future Networks and Communications/International Conference on Mobile Systems and Pervasive Computing (FNC/MobiSPC)*, volume 134 of *Procedia Computer Science*, pages 340–345. Elsevier, 2018.

[67] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *Journal of ACM*, 65(3):16:1–16:40, March 2018.

[68] Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *ACM SIGMOD Record*, 42(4):5–16, 2013.

[69] Amjad Qtaish and Kamsuriah Ahmad. Xancestor: An efficient mapping approach for storing and querying XML documents in relational database using path-based technique. *Knowledge-Based Systems*, 114:167–192, 2016.

[70] Richard M Royall. On finite population sampling theory under certain linear regression models. *Biometrika*, 57(2):377–387, 1970.

[71] Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM, 1978.

[72] Albrecht Schmidt, Florian Waas, Martin L. Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. Xmark: A benchmark for XML data management. In *The Proceedings of the VLDB Endowment (PVLDB)*, pages 974–985. Morgan Kaufmann, 2002.

[73] Juwei Shi, Jia Zou, Jiaheng Lu, Zhao Cao, Shiqiang Li, and Chen Wang. Mrtuner: A toolkit to enable holistic optimization for mapreduce jobs. *The Proceedings of the VLDB Endowment (PVLDB)*, 7(13):1319–1330, 2014.

[74] Rekha Singhal and Praveen Singh. Performance Assurance Model for Applications on SPARK Platform. In *Proceeding of the Technology Conference on Performance Evaluation and Benchmarking (TPCTC)*, pages 131–146. Springer, 2017.

[75] Ahmed A. Soror, Umar Farooq Minhas, Ashraf Aboulnaga, Kenneth Salem, Peter Kokosielis, and Sunil Kamath. Automatic virtual machine configuration for database workloads. In *International Conference on Management of Data (SIGMOD)*, pages 953–966. ACM, 2008.

[76] Jian Tan, Tieying Zhang, Feifei Li, Jie Chen, Qixing Zheng, Ping Zhang, Honglin Qiao, Yue Shi, Wei Cao, and Rui Zhang. ibtune: Individualized buffer tuning for large-scale cloud databases. *The Proceedings of the VLDB Endowment (PVLDB)*, 12(10):1221–1234, 2019.

[77] Rubén Tous, Anastasios Gounaris, Carlos Tripiana, Jordi Torres, Sergi Girona, Eduard Ayguadé, Jesús Labarta, Yolanda Becerra, David Carrera, and Mateo Valero. Spark deployment and performance evaluation on the marenostrum supercomputer. In *IEEE International Conference on Big Data (Big Data)*, pages 299–306. IEEE, 2015.

[78] Todd L Veldhuizen. Leapfrog triejoin: A simple, worst-case optimal join algorithm. *arXiv preprint arXiv:1210.0481*, 2012.

[79] Shivaram Venkataraman, Zongheng Yang, Michael J Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 363–378. USENIX Association, 2016.

[80] Shivaram Venkataraman, Zongheng Yang, Michael J. Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 363–378. USENIX Association, 2016.

[81] Guolu Wang, Jungang Xu, and Ben He. A novel method for tuning configuration parameters of spark based on machine learning. In *IEEE International Conference on High Performance Computing and Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 586–593. IEEE, 2016.

[82] Kewen Wang and Mohammad Maifi Hasan Khan. Performance prediction for apache spark platform. In *International Conference on High Performance Computing and Communications, International Symposium on Cyberspace Safety and Security, and International Conference on Embedded Software and Systems (HPCC/CSS/ICESS)*, pages 166–173. IEEE, 2015.

[83] Bifan Wei, Jun Liu, Jian Ma, Qinghua Zheng, Wei Zhang, and Boqin Feng. Motif-based hyponym relation extraction from wikipedia hyperlinks. *IEEE Transactions on Knowledge and Data Engineering*, 26(10):2507–2519, 2014.

[84] Thomas Weise. *Global Optimization Algorithms - Theory and Application*. Self-Published, 2009.

[85] Thomas Wirtz and Rong Ge. Improving mapreduce energy efficiency for computation intensive workloads. In *International Green Computing Conference and Workshops (IGCC)*, pages 1–8. IEEE Computer Society, 2011.

[86] Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural language engineering*, 11(2):207–238, 2005.

[87] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *The Proceedings of the VLDB Endowment (PVLDB)*, pages 82–94. IEEE Computer Society, 1981.

[88] Tao Ye and Shivkumar Kalyanaraman. A recursive random search algorithm for large-scale network parameter configuration. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 196–205. ACM, 2003.

[89] Zhibin Yu, Zhendong Bei, and Xuehai Qian. Datasize-Aware High Dimensional Configurations Auto-Tuning of In-Memory Cluster Computing. In *Proceeding of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 564–577. ACM, 2018.

[90] Chao Zhang, Jiaheng Lu, Pengfei Xu, and Yuxing Chen. Unibench: A benchmark for multi-model database management systems. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 7–23, 2018.

[91] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *International Conference on Management of Data (SIGMOD)*, pages 415–432. ACM, 2019.

[92] Junru Zhou and Hai Zhao. Head-driven phrase structure grammar parsing on penn treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2396–2408. Association for Computational Linguistics, 2019.

[93] Huchao Zhu, Huiqun Yu, Guisheng Fan, and Huaiying Sun. Mini-xml: An efficient mapping approach between XML and relational database. In *International Conference on Computer and Information Science (ICIS)*, pages 839–843. IEEE Computer Society, 2017.

[94] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning. In *Proceeding of the 8th ACM Symposium on Cloud Computing (SoCC)*, pages 338–350. ACM, 2017.